Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1982-10

Tactical motion analyzer (TMA).

McCorkle, Jack Woodward Jr.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/20370

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

DLEY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF. 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TACTICAL MOTION ANALYZER (TMA)

by

Jack Woodward McCorkle Jr.

October, 1982

Thesis Advisor: Alvin F. Andrus

Approved for public release, distribution unlimited

T205747

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Tactical Motion Analyzer (TMA)	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, October, 1982
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Jack Woodward McCorkle, Jr.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: October 1982
	13. NUMBER OF PAGES: 109
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report)
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Wargaming, Microcomputer Wargames, Motion Analyzer, Gaming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis is a two player, microcomputer controlled, tactical motion analyzer (TMA) of unit movements. TMA is designed to be used in either a manual or computer driven wargame. Written in the computer language Pascal, implementation is designed for the APPLE II computer. The program is contained on two, 5 inch floppy diskettes. Specific unit characteristics are entered into the database by the users. Users may then analyze unit movements using time-step simulation. Data transfer between computers is

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

done by the users physically exchanging diskettes. The game may be replayed or saved and continued at any time. The program logic and data manipulation are discussed in detail within the text of the thesis.

Approved for public release, distribution unlimited

Tactical Motion Analyzer (TMA)

by

Jack Woodward McCorkle Jr.
Lieutenant, United States Navy
B.S., United States Naval Academy, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
October, 1982

Thesis
M8193
C.1

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF. 93940

ABSTRACT

This thesis is a two player, microcomputer controlled, tactical motion analyzer (TMA) of unit movements. TMA is designed to be used in either a manual or computer driven wargame. Written in the computer language Pascal, implementation is designed for the APPLE II computer. The program is contained on two, 5 inch floppy diskettes. Specific unit characteristics are entered into the database by the users. Users may then analyze unit movements using time-step simulation. Data transfer between computers is done by the users physically exchanging diskettes. The game may be replayed or saved and continued at any time. The program logic and data manipulation are discussed in detail within the text of the thesis.

TABLE OF CONTENTS

I. INTRODUCTION 8

II. GOAL OF THE TMA PROGRAM 10

A. WARGAMES IN GENERAL 10

B. TMA IN GENERAL 13

III. THE PROGRAM 15

IV. MOTION ANALYZER 20

A. INITIALIZATION 20

B. MOTION ANALYSIS 22

1. Segment procedure MATHFUNC 22

2. Segment procedure MOTION 23

3. Segment procedure ADVANCE 24

4. Segment procedure CIRCLE 26

5. Procedure RANGE 26

6. Segment procedure CPA 27

7. Procedure INTERROGATE 31

8. Segment procedure MOVEMENT 31

9. Segment procedure MOVE2 32

10. Segment procedure INTERCEPT 33

11. THESIS0 library segment 33

12. Segment procedure RECENTER 34

13. Toggle the display 35

14. Terminating TMA 35

V. THE DATABASE 36

VI. THE RERUN OPTION 40

VII. THE USER'S MANUAL 41
VIII. AREAS FOR FUTURE ENHANCEMENTS 43
COMPUTER PROGRAM 45
LIST OF REFERENCES 107
BIBLIOGRAPHY 108
INITIAL DISTRIBUTION LIST 109

LIST OF FIGURES

1. Monitor before advance 25
2. Monitor after advance 25
3. Monitor with circle displayed 28
4. Monitor with range displayed 28
5. Monitor showing CPA 30
6. Monitor showing unit movement 30

I. INTRODUCTION

The purpose of this thesis is to provide a user friendly, two player, microcomputer controlled, tactical motion analyzer (TMA) of unit movements.

Since their advent, computers have been used by the military and civilian contractors alike in an attempt to simulate war environments. This was first done by the vacuum-tube computers to solve firing solutions for anti-aircraft guns, and later by high speed, mainframe computers to simulate complete battles and even war scenarios. Only recently has the technology been available to do similar applications on microcomputers. Specifically, memory chip density advances have to be the most significant contribution in this field.

The hardware to do large scale wargaming has been available for several years, but it is only now that software development is approaching the sophistication level so that non-computer enthusiasts can sit down at a terminal and successfully interact with a computer wargame. It is with this in mind that this thesis was written. A primary goal of TMA is to design human factors and ease of use into a game as it is written. This subject is discussed in depth in chapter III.

Although TMA is a complete program, there are many unexplored areas that could enhance it. Chapter VIII lists some suggestions for future expansion.

In order to use the TMA program, the following equipment is required: the TMA diskettes (two for each player), two Apple II computers, two disk drives and a monitor for each machine.

Prospective users of this program may include anyone interested in observing relative unit motion while interacting with a microcomputer. Since this program is an interactive, two person game, these individuals will be able to test strategies for maneuvering their units. Through playing the game and becoming familiar with some surface vessel tactics, it is hoped that this program will be of some educational benefit to U.S. Navy line officers.

II. GOAL OF THE TMA PROGRAM

A. WARGAMES IN GENERAL

The author has had some limited experience with U.S. Navy funded, commercially produced, interactive computer wargames. This experience is restricted to the Warfare Environment Simulator (WES). WES is a real-time man-interface discrete time step simulation for analytical wargaming applications in support of a broad range of research and development objectives. WES was produced by System Development Corporation (SDC) located in San Diego, California. WES appears to do what it's suppose to do, however, it also has some very difficult characteristics. Specifically, it is difficult to use. A variety of special commands and code words must be learned before playing the game. These are many in number and nonstandard compared to military nomenclature. In many cases numerous man-hours must be spent in becoming familiar with these terms which are most often germane only to the present system.

Many computer wargames are designed for implementation on large, fast and expensive mainframe computers. This very description implies a limitation for its users. Due to the high initial costs and maintenance expenses, general user availability is less than optimal. Additionally, software "experts" must be on hand to answer questions and help train system users. This makes the wargame expensive to run and maintain.

Another problem that is not restricted to military wargames, but to sophisticated software in general is lack of good documentation! Granted that software engineers who develop, design, write and test programs are gifted individuals, their ability to write user instructions for the layman is suspect. After spending months and sometimes years developing software packages it is understandable that these same people may lose perspective. There are two extremes to the documentation dilemma; too technical or too general. It seems as if the too technical examples were written by the software engineers while the authors of the too general documentation never really understood what the program was supposed to do.

Price notwithstanding, ease of use or user friendliness may be the major factor influencing acceptance of microcomputers and computer generated software in the future. Several articles have appeared in recent magazine issues of BYTE [Ref. 1] and PERSONAL COMPUTING [Ref. 2] dedicated to the subject of human factors design engineering in hardware and software systems. Basically they both say that as the general public begins to accept microcomputers into their daily routines, the transition to a computerized society will be made smoother by the software designers. They must design their programs with the general public in mind. Civilian contractors who supply wargames to the military have a head start on the rest of the computer software industry, but there is still a long way to go.

Another possible advantage in using microcomputers for wargaming is reducing the intimidation the user feels while sitting in a large computer center time sharing with a mainframe computer. Anxiety and trepidation on the behalf of the user may be kept to a minimum by using micros that incorporate user friendly software.

Two major disadvantages associated with using microcomputers for large scale wargaming are:

1. The lack of standardization in the microcomputer industry.
2. The inability of today's microcomputers to handle very large databases with the speed necessary to use them in "real time" applications.

Until an industry-wide standard is adopted purchasers of systems must ask themselves if what they're buying will be compatable with what they already have or what they may buy in the future.

The second disadvantage is not as serious as the lack of standardization since technological breakthroughs often occur. Today's 8-bit and 16-bit microprocessors will surely give way to tomorrow's 32 and 64-bit microprocessors. This increased size chip will allow vast amounts of online computer memory to be available for instant recall, thereby allowing increased database size and retrieval speed.

B. TMA IN GENERAL

The TMA program is a small scale attempt to correct many of the above listed deficiencies. The microcomputer has become a very powerful computing tool. This is due primarily to technological increases in the areas of silicon chip memory densities and larger, more sophisticated microprocessors with faster cycle times. The microcomputer's lower cost and high reliability make it possible to increase overall availability of computer wargames as well as computers in general.

The major goal of the TMA program is to demonstrate some user friendly software implementation techniques while exploring the capabilities of microcomputers in wargaming. In addition to being informative and instructive in nature, this program is intended to be painless to use and fun to watch. This is made possible by:

1. Programming the game in the computer language Pascal.
2. Using the high resolution graphics capability of the Apple II computer.

Pascal was chosen as a programming language over BASIC and FORTRAN because of its highly structured nature. This structuring allows, if not requires, modular design and makes it very legible and comprehensible for large programs. Changes and modifications can be made several times faster than in unstructured languages. Also, Ada, which is soon to be required of all contractors who want to do business with the U.S. Department of Defense, can be thought of as an

enhancement of Pascal. Pascal comes in many versions. The version used for TMA is Apple Pascal which incorporates UCSD Pascal and Apple extensions for graphics, sound, and paddles. Of these extensions TMA uses graphics and sound.

The high resolution graphics capability of the Apple II computer makes it possible to produce an aesthetically pleasing graphical presentation of the actual events as they happen during the simulation. This could be expanded upon by having a high resolution color monitor instead of a green or black and white screen. Visual representation prompts faster feedback from the user. He can quickly scan and understand each tactical situation as it occurs. The more computer graphics are used to represent what is actually happening, the more useful the wargame will be to the user.

III. THE PROGRAM

TMA is an acronym standing for Tactical Motion Analyzer. The main program is comprised of three separate subprograms; the database, the motion analyzer and a fast-time replay of the motion analyzer.

The database subprogram is designed to be interactive in nature. Menus are used throughout the program that offer the user a selection of specific choices that may be made. Once an appropriate choice is made, that segment of the program becomes active. If the user makes a mistake or changes his mind about a selection, the program will allow the user to regress to the previous level. When an input is required the computer will prompt the user. If an unexpected input is received the computer will beep, the input will be erased from the screen and the user will again be prompted for input. This will not prevent erroneous input from getting into the database, but will prevent the user from entering a number if a letter is expected and vise versa.

The database portion of the program allows the user to develop and save various parameters of specific units. These parameters are entered by the user and then saved as a file on the game diskette. This information is then available throughout the program using random access. There is a default database on the TMA1 diskette consisting of ten units

(specifically surface vessels). The user can use the default database or devise his own. Each game diskette has room for at least ten separate user database files. Input required by the database for each unit is as follows:

1. Class
2. Name
3. Hull number
4. Initial Course
5. Initial Speed
6. Position (X-Y coordinates)
7. Maximum unit speed
8. Number of surface-to-surface missiles (SSM) on board
9. SSM speed
10. Number of surface-to-air (SAM) missles on board
11. SAM speed

Course and speed are the only parameters which may be altered before each move.

Each user database must have a different name. When the diskette becomes full, the program will let the user know by printing an appropriate message on the screen. More room can be made available on the diskette by removing an unnecessary datafile.

The database subprogram is completely selfcontained. While the user is interacting with this section of the program, no data is transferred between computers.

The second subprogram of TMA deals with the mathematical manipulation and graphical display of the program. All calculations in TMA are deterministic. Motion for each unit is computed from three inputs provided by the user; course, speed and time. Course and speed are maintained in the database. A time increment is entered before every time-step.

Geometry and trigonometry are used to derive all unit motion. The calculations are first computed for the user's side, then the program automatically communicates with the opponent's program and retrieves the necessary data to plot the updated positions of his units. A history of all position changes are saved on the diskette for later use by the program. Communication between computers is handled by the user. The program will stop at the appropriate time and request the players to exchange program diskettes. This exchange allows each player to view the moves made by the opponent. If for some reason this exchange is not made, the program will continue to run, however, only the moves made locally will be displayed.

Between moves the user can study each unit's situation. The user has the following options:

1. Draw a circle of a specified range.
2. Compute the range and bearing between units.
3. Determine closest point of approach (CPA) between units.
4. Calculate intercepts between units.
5. Move individual units around the screen.
6. Advance to the next time-step.

A more specific description and explanation of how to use the program follows in the Appendix in the form of a user's manual.

The graphical representation of TMA is oriented in U.S. Navy standard. North is 360 or 000 degrees and corresponds to the top of the screen. East is 090 degrees and is the right side of the screen as the user views it. Bearings increase to the right (clockwise). The graphic screen of the

Apple computer is not square but is actually a rectangle that measures 280 dots along the X-axis (horizontally) and 192 dots along the Y-axis (vertically). This rectangular display has some inherent disadvantages. The disadvantage most noticable to the user will be that circles drawn on the display will appear as ellipses. Also, identical distances drawn along both axes will show on the screen as having different length.

The graphics screen of TMA, as viewed by the user, will consist of a 100 by 100 mile square grid. Each dot will represent one mile. The lower left-hand corner will be the origin. The horizontal and vertical axes correspond to the X and Y axes, respectively. These square coordinates will be linearly transformed, within the mechanics of the program, into the rectangular grid that the Apple computer uses. All numerical output to the user will again be transformed to the more familiar 100 by 100 grid.

This method of graphic display was chosen by the author to ensure a one-to-one correspondence between the user's grid coordinates and that of the machine. In retrospect, this may not be as important as originally envisioned. Consequently, this phenomenon is one that may be corrected by changes to the program code.

Calculations are performed using real numbers, which on the Apple Computer have precision to seven decimal digits. However, displaying these positions on the screen require rounding off each position to its integer equivalent. The

impact of this induced rounding error is varied depending on what scale the user is viewing the screen. For example, the initial scale is 1 dot per mile. In this scale the round off error may account for a screen positioning error of up to one mile.

The final section of TMA is the fast-time replay subprogram. When Rerun is invoked by the user, the screen will clear and display the game situation as it was before the first move of the game. During this phase of the program all user options are active just as before. However, when the user elects to advance to the next time step, instead of computing new positions, the computer will read the next set of coordinates from it's history file and display them on the screen. This gives the user the ability to observe a lengthy series of moves condensed into a relatively short time. This is an ideal time for the user to critique his own tactical decisions and to observe those of his opponent. The Rerun subprogram is completely selfcontained, that is, no interaction takes place between computers. Each terminal is a stand alone system at this time requiring no exchanging of diskettes.

IV. MOTION ANALYZER

A. INITIALIZATION

TMA is a large program as microcomputers go. The text version is over 3000 lines of Pascal code. The compiled length equates to about 40,000 bytes. This does not include the library routines that are used for graphics and trigonometric functions, nor memory that is set aside for variables. Although the Apple II is a 64,000 byte machine, it cannot store the Pascal language, TMA, and the library routines in memory at the same time. Pascal, however, is such a flexible language that it allows the programmer to divide programs into smaller segments that are only loaded into memory when needed. Only the 'main' program need reside in memory at all time.

The 'main' program is loaded into memory automatically when the machine is turned on, provided the TMA diskettes are in the disk drives. The 'main' program has two important functions:

1. The declaration of all global variables used in TMA.
2. The job of 'program director'.

Many variables used in TMA are global. The record which contains all database information, datafile names, and flag names are needed in various segments throughout TMA. In order to pass the values of these variables between program segments, they must be declared in the main program. Unfortunately, each global variable consumes precious memory.

The main program acts as program director by checking the status of each flag variable. Throughout the program various flags are set. These flags are boolean variables that have values of 'true' or 'false'. These flags are changed as the user progresses through the program. .When the user leaves a segment, TMA always returns to the main program to check the flags. The main program determines which segment is to be loaded into memory based on present flag status. This loading and unloading of segments into the memory is virtually invisible to the user. However, program response time is slowed by frequent access to the disk drives.

Once the main program is loaded, execution begins. TMA reads a file on the TMA1 diskette called SYSTEM.MISCINFO. This file contains the screen control characters for each computer. Using SYSTEM.MISCINFO allows the programmer to perform general video functions that are terminal independent. These functions include clearing the screen and positioning the cursor.

An initialization segment (TMAINIT) is loaded next. TMAINIT sets certain variables, arrays, and all flags to their initial values. This segment also asks the user if he wants to resume an old game. If the user responds affirmatively, the proper flag is set and the program transfers to the RESUME segment. The RESUME segment reads data from the last move of a previous game from five different disk files on the diskette TMA2. These five disk files are then removed

from the diskette. If the user opts to begin a new game, the motion analyzer subprogram is invoked by selecting the MOTION ANALYSIS PROGRAM option. This subprogram is the bulk of TMA.

B. MOTION ANALYSIS

1. Segment procedure MATHFUNC

The procedure MATHFUNC marks the actual beginning of the MOTION ANALYSIS PROGRAM. This procedure has several related functions.

MATHFUNC first asks the user the name of the diskfile containing the database for the game. The user enters the filename, and the program locates and reads that diskfile. Should the user misspell or make a mistake typing the filename, the program will continue to prompt the user until a correct filename is entered. The program, having read the desired datafile, computes each target's initial position and displays them on the monitor. This gives the user an opportunity to visually check the information in the database. The user must press the escape key (ESC) on the keyboard to continue. At this point, the user has two options. The user may elect to continue the game, or return to the database segment to make changes to the database. The database segment is discussed in detail in chapter 5.

If the user decides to continue then several events occur. TMA opens two historical datafiles on TMA1 (ORNGOLD.DATA and BLUEOLD.DATA). These datafiles maintain unit positioning data for use during the RERUN segment of TMA. Specifically,

each file contains an X and Y coordinate for each unit and the time increment for each game move. One additional disk file is initialized also (BLUENOW.DATA). This file, on TMA2, maintains the present positioning data for the user's units.

At this time TMA ceases to be a self-contained program and the user is informed that data transfer between players is necessary. Instructions are displayed on the monitor and the user need only follow them. These instructions request the user to remove the TMA2 diskette from drive #2 and exchange it with his opponent's TMA2 diskette, then place this TMA2 diskette in drive #2. As mentioned above, a datafile (BLUENOW.DATA) is maintained on TMA2 that contains current positioning data. When these diskettes are exchanged, current positioning data is also exchanged. Both sides now have current positioning data for themselves and their opponents. The MATHFUNC segment terminates, sets the proper flags, and transfers program control back to the main program.

2. Segment procedure MOTION

The main program transfers control of TMA over to the procedure MOTION. This procedure performs the actual motion analysis requested by the user. The user has the following options:

1. Advance to next time-step.
2. Draw range circle around a specified unit.
3. Compute range between units.
4. Determine CPA between units.
5. Determine Course/Speed of units.
6. Check/Change movements of a unit.
7. Compute intercept between units.

8. Clear the display.
9. Downscale display.
10. Upscale display.
11. Recenter the display on a specified unit.
12. Toggle between text and graphic page.
13. Terminate the game.

3. Segment procedure ADVANCE

The Advance option transfers control of TMA to the ADVANCE procedure. Based on a time step entered by the user, the program calculates new X, Y-coordinates for all units. The following formulae are used:

X' = X + (SPEED/60 x TIME x 2.8 x SIN(COURSE))

Y' = Y + (SPEED/60 x TIME x 1.92 x COS(COURSE))

X', Y' represent the new X and Y positions respectively. SPEED is the unit's speed in nautical miles per hour, the 60 converts it to minutes. TIME is the time-step entered by the user in minutes. The numbers 2.8 and 1.92 are the number of graphical dots per nautical mile in each plane X, Y. The trigonometric functions SIN and COS determine the X and Y components of the unit's velocity.

Before the actual computations are performed, however, the user is given the option of viewing and changing the present course and speed of each unit. This way minor changes may be made to the database without returning to the database procedure.

A pictoral representation of units before and after they advance is shown in figures 1 and 2 on the following page. Figure 1 shows three units as they may appear before being advanced. Figure 2 shows the units after they have

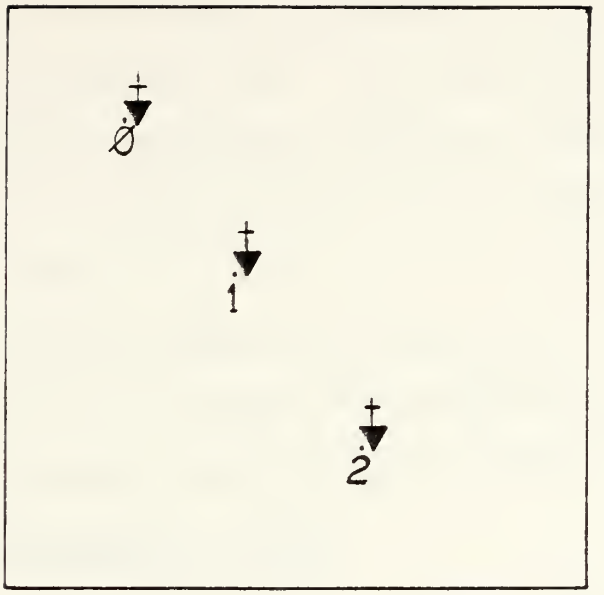

Figure 1: Monitor before advance

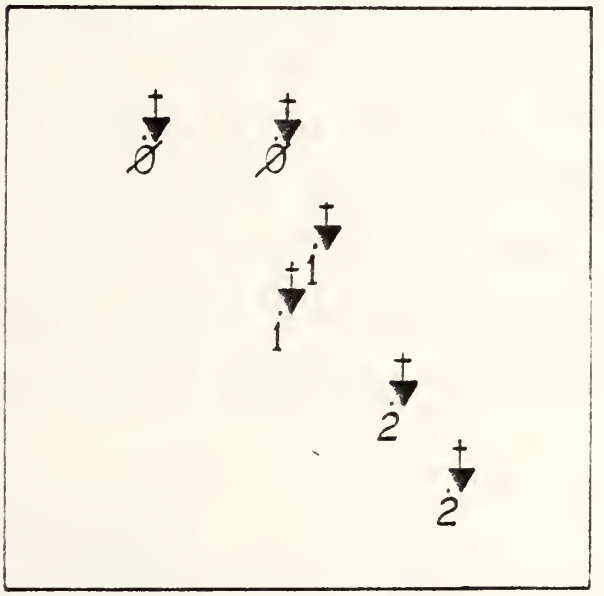

Figure 2: Monitor after advance

advanced. Both the before and after positions will be displayed until the user clears the screen or changes scale.

After the new X and Y coordinates are calculated, they are saved on the diskette TMA2. Once again the user goes through the diskette exchanging procedure with his opponent. These coordinates are then saved in the historical datafiles on the diskette TMA1.

The ADVANCE procedure then transfers control back to the MOTION procedure. The new unit positions are displayed and the user may again select from the various options.

4. Segment procedure CIRCLE

This procedure allows the user to draw circles of a specified range around any unit. The TMA program prompts the user for the circle's origin and a radius in nautical miles. Due to the graphical limitations of the APPLE II computer, these circles must be drawn one dot at a time. The circles will always appear elliptical vice circular. This is because of the difference in the number of dots per mile along the horizontal axis and the vertical axis. Figure 3 shows the display after a circle is drawn around unit number one. After the circle is completed, control of TMA is returned to MOTION.

5. Procedure RANGE

Procedure RANGE is incorporated into the MOTION segment procedure since it is relatively small. RANGE calculates the distance between any two units. The program prompts the user

for the unit numbers desired, target 1 and 2. A line is drawn between the units, and the range is displayed in nautical miles. Range is calculated using the square root of the sum of the squares.

DIST = SQRT(((X2-X1/2.8)^2 + ((Y2-Y1/1.92)^2)

where,

X1, Y1 is the present position of target 1. X2, Y2 is the present position of target 2. The factors 2.8 and 1.92 transform the units from dots per nautical mile to nautical miles. The term '^2' means raised to the second power. SQRT implies take the square root.

Figure 4 shows the display after the range between unit zero and unit one has been calculated. After displaying the calculated range between targets, program control is returned to the MOTION segment.

6. Segment procedure CPA

The CPA procedure calculates the closest point of approach between two designated units. Computations are based on present course and speed.

CPA derives all information, course and speed, from the two most recent unit positions. Obviously, if no advances have been made, course and speed can not be determined. CPA will inform the user that not enough data exists to calculate CPA.

Assuming at least one time-step advance has been made, CPA is calculated as follows:

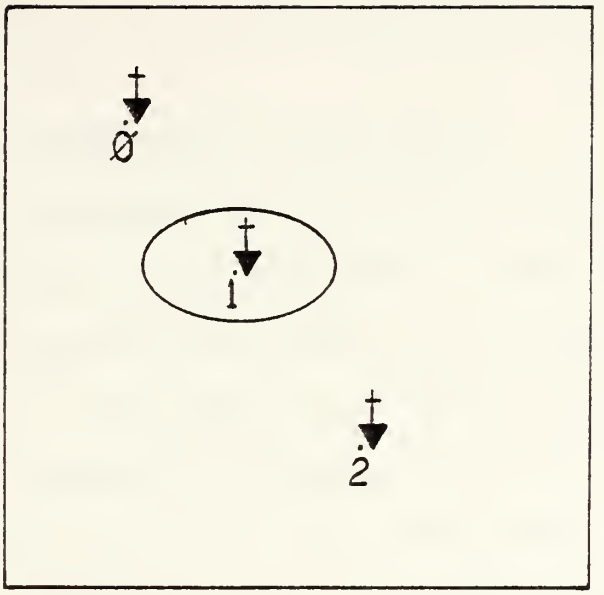

Figure 3: Monitor with circle displayed

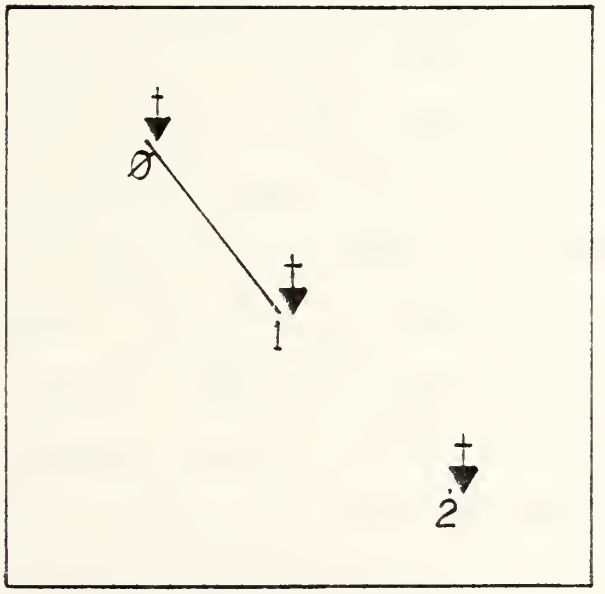

Figure 4: Monitor with range displayed

The two most recent positions of all units are maintained in a global array throughout TMA. Based on these positions the course of any unit can be calculated using the arctangent function.

COURSE = ARCTANGENT((DX/2.8)/(DY/1.92))

DX is the distance moved in the X direction and DY is the distance moved in the Y direction. Once again these distances are translated from dots per nautical mile to nautical miles. A problem occurs when DY is zero or close to zero. By checking this first the program can avoid 'division by zero' errors. If DY is zero the course must be either 090 or 270 degrees. Again this can be determined by the sign of DX. If DX is positive, for example, the unit is traveling 270 degrees. Unit speed is calculated in a two stage process. Distance is calculated first using the same method described in RANGE above. Speed is derived from this distance divided by the length of time of the last time-step.

Knowing the course and speed of the units the program uses the X and Y components of velocity to compute the time of minimum distance. This is the time of CPA. Once CPA time is known, the relative future positions of each unit is calculated and displayed. Figure 5 shows the screen after TMA has calculated CPA between unit zero and unit one. The end point of the displayed lines is the relative unit positions at CPA. Finally, the program informs the user of the number of minutes until CPA, and the range and bearing between units at CPA.

Control is then returned to the MOTION segment.

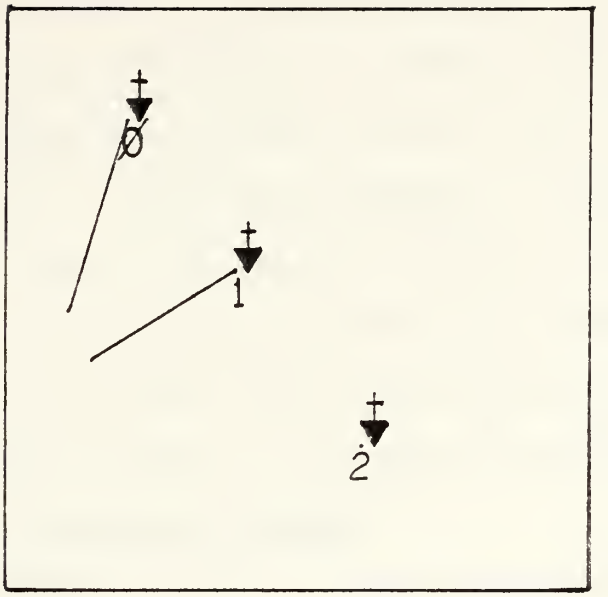

Figure 5: Monitor showing CPA

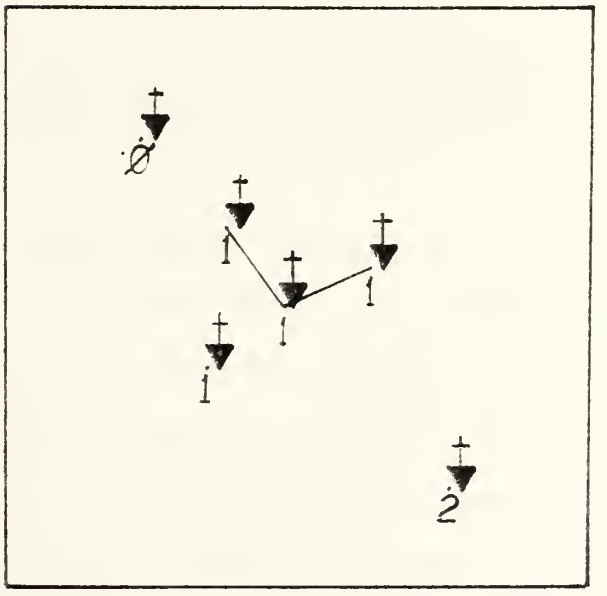

Figure 6: Monitor showing unit movement

7. Procedure INTERROGATE

This procedure determines the course and speed of any unit. As in CPA above, this procedure uses the two most recent positions of the target to determine its course and speed. The program will inform the user that it does not have enough data. If the user attempts to invoke this procedure after at least one time-step advance, then the unit's course and speed are calculated exactly as described above in CPA. After the course and speed has been determined and displayed, program control is returned to MOTION.

8. Segment procedure MOVEMENT

The MOVEMENT procedure moves individual units around the grid before actually making the changes permanent in the database. Three types of movement are possible for the player's units and one type for the opponent's units:

1. Movement using course, speed and time.
2. Move your unit to any X,Y coordinate.
3. Move your unit a bearing and distance.
4. Move your opponents units based on their present course and speed.

Option one prompts the user for the unit number, desired course, speed and time. The future position is calculated using SIN and COS functions, and then displayed on the monitor. Option two prompts the user for a unit number and an X and Y coordinate, which is then displayed on the monitor. Option three prompts the user for a unit number, bearing and distance. This is transformed into an X and Y coordinate, then displayed on the monitor. Option four is

not directly visible to the user, nonetheless, it is there. If the user wishes to move an opponent's unit, option four is automatically invoked. Option four transfers control to segment procedure MOVE2.

All movements allow the user the option trace-on or trace-off. Trace on draws a line from the old position to the new position. Trace off will not. After making a move the program asks the user if it should incorporate the move into the database for the next time-step. If the user answers affirmatively, the new course and speed for the unit moved will be entered.

Figure 6 shows three different movements of unit number one. Note that two movements are with the trace-on and one is with the trace-off. All unit movements will remain visible on the monitor until the user clears the screen.

The MOVEMENT segment can only be terminated by the user. Once terminated, program control is returned to MOTION.

9. Segment procedure MOVE2

This segment is invoked only when the user wants to move an opponent's unit. Since the user has no control over the course and speed of these units, all movements are based on the individual unit's actual course and speed. Course and speed are computed exactly as discussed above in segment procedure CPA. Once again at least one time-step advance must have been made to compute the unit's course and speed. Assuming the data is available, the user is asked for a

time-step increment to see the unit's future position. The only other option allowed is the trace-on or trace-off. This also works exactly as described above in the MOVEMENT segment.

Completion on the MOVE2 segment returns control of the program to the MOVEMENT segment.

10. Segment procedure INTERCEPT

This procedure computes the course and speed required to intercept any unit from any user's unit. Three inputs are required by the user. Input one is the unit to be intercepted. The second input is the unit number of the intercepting unit. Note, this unit must be maneuverable by the user. The final input is the desired time of intercept (minutes). Once again at least one time-step advance must be made to compute an intercept. The time supplied by the user will determine the point of interception, assuming the data is available to compute course and speed of the unit to be intercepted. An intercept course is computed and speed is calculated based on the present position of the intercepting unit and the point and time of interception. This course and speed is displayed on the monitor and the user is asked if he wants to make this course and speed change to the database for the intercepting unit.

When completed, control is returned to the MOTION segment.

11. THESIS0 library segment

The downscale and upscale options are contained in the SYSTEM.LIBRARY file on diskette TMA1. These options allow the user to change the scale of the display at any time.

The left arrow key (←) controls the downscale function. This function clears the display and then redisplays all units at a scale one half the previous value. For example, if the scale is 10 dots per nautical mile, it becomes 20 dots per nautical mile after a downscale.

The right arrow key (→) controls the upscale function. It is the exact inverse of the downscale function. If the scale is 10 dots per nautical mile, it becomes 5 dots per nautical mile after an upscale.

Neither downscale nor upscale have any affect on the actual positions of the unit's coordinates used for calculations. These positions are kept in global arrays that are used throughout the program. The present scale is displayed at the top of the screen at all times. The scale shows the relative horizontal distance of ten miles. The user should note that the same vertical distance appears shorter because the number of dots per nautical mile is less. Automatic scaling is not available, the scale will not change unless the user exercises his options.

12. Segment procedure RECENTER

This procedure belongs in the THESIS0 library, however, was made into its own segment because of memory constraints. RECENTER, as its name implies, recenters the display. The space bar activates this procedure. The user may recenter the display on any unit. This becomes very important as the game advances and the targets move.

Along the top of the display, next to the scale, are the center coordinates. These coordinates are initially (50,50) which is the center of the screen. As the user recenters, the top of the display always indicates the current screen center.

After recentering, the control of the program returns to MOTION.

13. Toggle the display

The APPLE II computer has two separate screens, the text screen and graphics screen. The TMA program tries to do a good job of anticipating the users needs by switching between text and graphics. However, the user has the option of switching between text and graphics on his own by using the escape (ESC) key.

14. Terminating TMA

The final option available to the user is to terminate the program. TMA displays the same menu as was shown at the beginning of the game. If the user selects the TERMINATE PROGRAM option, the user is asked if he wants to save the game. This option should be selected if the user wishes to return later to finish the game. This option creates five different datafiles on diskette TMA2 for future use. When the user wishes to continue the game, answer affirmatively to that question and assure that TMA2 is in drive #2. The program will load the appropriate data, initialize the proper variables and return the user to the MOTION segment procedure as if he had never left.

V. THE DATABASE

The database subprogram is invoked by selecting the 'BUILD OR MODIFY DATABASE' option at the start of the game. The database uses random access disk files to build and maintain a record of specific unit parameters. Modifications to an existing database are quickly made due to the random access structure.

Initially, the program asks the user to enter a file name for the database. After the file name is entered, two possibilities exist:

1. The file name already exists on the game diskette.
2. The file name does not exist on the diskette.

If the name is presently on the game diskette, the program will read that file into the computer. That information will then be available for viewing and editing. If the file name is not on the diskette, the program will ask the user if he wants to start a new file. A 'no' response will cause the computer to again ask the user for a file name. A 'yes' response will prompt the program to ask how many records are needed for the new datafile. If the number of records is not known, a best guess is required. The number of records is necessary to define disk space for random access. However, if the user decides to extend the number of records the program will automatically extend the size of the datafile. The program

will notify the user should the diskette become too full to extend the datafile. To save disk space it is generally better to initially under estimate the number of records than to over estimate.

At this point the user will choose from the following options:

1. View the present database.
2. Change the present database.
3. Build a new game database.

The option chosen will determine how 'deep' into the database the user will go. Options one or two allow the user access only to the upper level of the database. Option three allows access to both the upper and lower levels. The upper level consists of the following:

1. Class:
2. Hull number:
3. Maximum speed:
4. Number SAM:
5. SAM speed:
6. Number SSM:
7. SSS speed:

This option is recommended when the user is entering unit data, but is not ready to start a game.

The lower level consists of the following:

1. Name:
2. Course:
3. Speed:
4. X Coordinate:
5. Y Coordinate:

This option is recommended when the user is ready to start the game. These inputs determine the initial starting position and speed for each unit.

After choosing which level of the database to enter, the program clears the screen and displays a prompt line along the top. This prompt line affords the user the following options:

1. Help
2. View
3. Change
4. Next
5. File
6. Return
7. Quit

The Help option explains each option. The View option displays each record for the users inspection. No changes may be made. The Change option displays each record and prompts the user for input. Changes are made respective to the level the user is in. The Next option displays the next record in the file. The File option saves the present database to the diskette and asks the user for another filename. The Return option allows the user to change levels within the database. Quit saves the present database to diskette and terminates the database session of TMA.

Data entry into the database has been designed to be as painless as possible. The program displays the present contents of each field, when changes are required, on the top half of the screen. The user is prompted on the bottom half of the screen for input, one field at a time. If a restriction exists on the range of that input, that range is displayed as well. Any field may be left unchanged by either retyping that data or by using the RETURN key. The return key will

automatically duplicate the present data into the changed database for that field.

Each record in the database is numbered by the computer. These numbers begin at zero and continue sequentially to N - 1, where N is the number of records in the database. Any record in the database may be viewed or changed at any time by entering that record number. Random access will minimize the time required to make minor changes to one or a few specific records.

There exists on the TMA1 diskette two predefined databases, BLUEMASTER and BLUEGAME or ORANGEMASTER and ORANGEGAME depending on which side the user happens to be. These databases contain ten ships with characteristics from JANE'S FIGHTING SHIPS. They are provided for the user to use as he wishes. The user may modify these files or use them as they are in lieu of building his own database. It is recommended the user view these database files to get an idea of the type of input expected by the program.

VI. THE RERUN OPTION

The RERUN subprogram is invoked by selecting the 'RERUN MOTION ANALYSIS PROGRAM' option at either the start of the game or after terminating a game. This option allows the user to view a fast-time replay of the most recent session.

As mentioned in a previous chapter, all unit positioning data is saved in datafiles during the game on diskette TMA1. The RERUN subprogram resets these datafiles and sequentially displays the unit positions. All TMA motion analysis options as described in chapter IV are available in RERUN.

There is one difference between the RERUN and MOTION ANALYSIS subprograms. That difference is in the segment procedure ADVANCE. The RERUN version is a reconstruction of past events. When the advance option is selected, the user is not given an opportunity to examine or change the inputs to the database (unit course and speed). The ADVANCE segment only reads the next positioning data from the diskette files and displays it on the monitor. As always the user can analyze all unit motion. TMA will terminate when all available data has been displayed.

Having a RERUN option gives the user a capability to repeatedly view and review all the moves of a previous session. It is worth noting that during this subprogram TMA is self-contained. No interaction between players is necessary since all positioning data is stored on the diskette.

VII. THE USER'S MANUAL

The following is a step by step instruction guide to facilitate the running the TMA by an unexperienced user.

A. EQUIPMENT NEEDED

1. Two game diskettes marked TMA1 and TMA2 for each player.
2. Two APPLE II computers with 64,000 bytes of RAM (random access memory).
3. Two disk drives and one monitor per computer.

B. OPTIONAL EQUIPMENT

1. Videx 80 column board with soft-switch installed in APPLE's slot #3.
2. Color monitor.

C. GETTING STARTED

1. Turn on the monitor.
2. Insert TMA1 into disk drive #1 and TMA2 into drive #2. Remember to close the doors!
3. Turn on the APPLE. The power switch is located on the left side of the APPLE in the back.
4. The TMA program is automatically loaded.

D. TROUBLESHOOTING POSSIBLE PROBLEMS

1. No power--Ensure all hardware is properly connected. If the user is unfamiliar with the equipment, seek out an experienced user.
2. TMA does not load.
   a. Check that the diskettes TMA1 and TMA2 are in the correct disk drives.

b. Ensure that the APPLE II is configured as listed above.

c. Attempt the startup procedures on the second APPLE. If not successful, the diskettes may be defective, use the backup diskettes.

d. Seek assistance from an experienced user.

At this point, if the user has been successful, TMA is loaded and ready to run. Chapters 4, 5, and 6 explain in detail how TMA works and what inputs are required from the user. TMA is menu driven and designed to be user friendly. The author hopes the users enjoy it.

VIII. AREAS FOR FUTURE ENHANCEMENTS

TMA is a small scale model of what a microcomputer, interactive wargame can do. As such there is much room for program enhancement. One reason the author chose Pascal as the program language is the readability of the code. It should be relatively easy for a user familiar with Pascal to make changes to the program.

The design of TMA has significantly changed from initial logic to final product. The incorporation of various changes has resulted in a program that is far from optimal. The present design of TMA uses all the segments and library routines that Apple Pascal can handle. Therefore, any changes other than very small ones, will require the user to somehow condense the present code.

The present inefficiency of the program is illustrated by the following example. The procedures that compute a unit's course and speed are repeated in four different segment procedures. If this were incorporated into a library it would only be written once and could be removed from each segment. There are several occurances of similar situations.

Another method of streamlining the program is by chaining programs together. The built-in library function CHAINSTUFF provides the necessary machinery to chain together as many programs as can fit on a diskette. To use CHAINSTUFF on TMA

would require the reduction of libraries or segments presently used by at least one. With chaining the author can create many separate programs that appear to the user as a single large program. A user may wish to supplement TMA with some additional motion analyzing options. Additions such as this may easily be appended to the present program. If lack of memory becomes a problem some of the above enhancements will be required also.

A final area that could be enhanced is the individual algorithms themselves. The algorithms were written for clearity not efficiency. Many of the procedures and functions of TMA could be changed to conserve memory.

COMPUTER PROGRAM

```
(*$N+*)
(*$S++*)
(*$V-*)
PROGRAM TMA; ( THESIS )

USES
    TURTLEGRAPHICS,TRANSCEND,GETCRT,THESIS0;

CONST
    PI=3.141592654;

TYPE
     SHIP=RECORD
               CLASS:        STRING[10];
               NAME:         STRING[10];
               HULLNO:       STRING[8];
               SCOURSE:      INTEGER;
               SSPEED:       INTEGER;
               XPOS:         INTEGER;
               YPOS:         INTEGER;
               MAXSSPD:      INTEGER;
               NSAM:         INTEGER;
               SAMSPD:       INTEGER;
               NSSM:         INTEGER;
               SSMSPD:       INTEGER
               END;


VAR CH: CHAR;
    TEXT,BLFLAG,RCTRFLAG,CONTINUE,LASTCHANGE,BUILDING,OBCALLED: BOOLEAN;
    MMFLAG,RRFLAG,TFLAG1,TFLAG2,TFLAG3,TFLAG4,TFLAG5,TFLAG6,TFLAG7: BOOLEAN;
    STOPFLAG,GRAFFLAG,INITFLAG,QUITFLAG,OLDFLAG,CASE0,CASE1,CASE2: BOOLEAN;
    CASE0A,OLDGAME,ICPTFLAG: BOOLEAN;
    OKSET: SETOFCHAR;
    DATAFILE:   FILE OF SHIP;
    BLUHIST,BLUGAME,ORNHIST,ORNGAME: FILE OF REAL;
    ATIME:  FILE OF INTEGER;
    MAXSHIP,TIME,ORANGEMOVE,BLUEMOVE,MTARGET,DELTAT,RECNUM:     INTEGER;
    FILENAME:   STRING[15];
    GCUS,GSPD: PACKED ARRAY[1..10] OF REAL;
    OLDBXY,OLDOXY: PACKED ARRAY[1..2,1..10]OF REAL;

 (*$I#5:TMAINIT.TEXT*)
 (*$I#5:TMA081.TEXT*)
 (*$I#5:RESUME.TEXT*)
```

```
(*$I#5:MATH.TEXT*)
(*$I#5:MOTION.TEXT*)
(*$I#5:ADVANCE.TEXT*)
(*$I#5:CIRCLE.TEXT*)
(*$I#5:CPA.TEXT*)
(*$I#5:MOVEMENT.TEXT*)
(*$I#5:MOVE2.TEXT*)
(*$I#5:RECENTER.TEXT*)
(*$I#5:BLINE.TEXT*)
(*$I#5:TMAFINIT.TEXT*)

BEGIN  ( TMA )
(*$R GETCRT*)
  GETCRTINFO;
  OBCALLED:=FALSE;
  TMAINIT;
  REPEAT
    IF OLDFLAG THEN RESUME;  ( read in old positions  )
    IF INITFLAG THEN TMAINIT;
    IF CASE0 THEN MATHFUNC;
    IF (CASE0A) AND (NOT RCTRFLAG) AND (NOT BLFLAG) THEN MATHFUNC;
    IF CASE1 THEN RECENTER;
    IF CASE2 THEN DATABASE;
    IF (CONTINUE) AND (NOT BLFLAG) THEN MOTION;
    IF (BLFLAG) AND (NOT RCTRFLAG) THEN BOTTOMLINE;
    IF (TFLAG1) AND (NOT BLFLAG) AND (NOT RCTRFLAG) THEN MOVEMENT;
    IF MMFLAG THEN MOVE2;
    IF (TFLAG2) AND (NOT BLFLAG) AND (NOT RCTRFLAG) THEN MOTION;
    IF TFLAG3 THEN CPA;
    IF TFLAG4 THEN MOTION;
    IF (TFLAG5) AND (NOT RRFLAG) THEN ADVANCE;
    IF (TFLAG5) AND (RRFLAG) THEN CIRCLE;
    IF (TFLAG6) AND (NOT BLFLAG) THEN MOTION;
    IF TFLAG7 THEN CIRCLE;
    IF RCTRFLAG THEN RECENTER;
    IF QUITFLAG THEN TMAFINIT;
  UNTIL STOPFLAG;
END. ( TMA )

SEGMENT PROCEDURE TMAINIT;
VAR GFLAG,GFLAG1: CHAR;
    LEVEL0: SETOFCHAR;
PROCEDURE PRESTART;
BEGIN
  OKSET:=['Y','y','N','n'];
  GOTOXY(0,0);CRT(ERASEOS);
  WRITELN;
  WRITELN('  MEMORY AVAILABLE IN WORDS: ',MEMAVAIL);
  WRITELN;
  WRITELN('  MOTION ANALYSIS PROGRAM OPTIONS : ');
```

```
  WRITELN;
  WRITELN('  DO YOU WISH TO RESUME AN OLDGAME (Y/N)? ');
  WRITELN;WRITELN;
  WRITE('  ');    GFLAG1:=GETCHAR(OKSET+[CHR(13)]);
  CRT(ERASEOS); CRT(ERASEOS);
  IF GFLAG1 IN ['Y','y'] THEN OLDFLAG:=TRUE;
END; ( PRESTART )

PROCEDURE START;
BEGIN
  GOTOXY(0,0);CRT(ERASEOS);
  WRITELN;
  WRITELN('  MEMORY AVAILABLE IN WORDS: ',MEMAVAIL);
  WRITELN;
  WRITELN('  MOTION ANALYSIS PROGRAM OPTIONS : ');
  WRITELN(' ');
  WRITELN('  0  Motion Analysis Program');
  WRITELN('  1  Rerun Motion Analysis Program');
  WRITELN('  2  Build or Modify Data Base');
  WRITELN('  3  Terminate Program');
  WRITELN;
  WRITE('  ');    GFLAG:=GETCHAR(LEVEL0);
  CRT(ERASEOS); CRT(ERASEOS)
END; ( START )


PROCEDURE SETFLAGS;
BEGIN
  CONTINUE:=FALSE;(  trf to motion from math     )
  DBCALLED:=FALSE;(  set to true if database was called )
  INITFLAG:=FALSE;(  calls tmainit seg.procedure  )
  OLDFLAG:=FALSE; (  set to true if player wishes to resume old game )
  OLDGAME:=FALSE; (  set to true if this is a resumption of an old game)
  BLFLAG:=FALSE;  (  trf to bottomline )
  RRFLAG:=FALSE;  (  trf to rerun )
  MMFLAG:=FALSE;  (  trf to move2 from movement )
  TFLAG1:=FALSE;  (  trf to movement from motion )
  TFLAG2:=FALSE;  (  trf to motion from movement )
  TFLAG3:=FALSE;  (  trf to cpa from motion  )
  TFLAG4:=FALSE;  (  trf to motion from cpa  )
  TFLAG5:=FALSE;  (  trf to advance from motion  )
  TFLAG6:=FALSE;  (  trf to motion from advance  )
  TFLAG7:=FALSE;  (  trf to circle from motion   )
  ICPTFLAG:=FALSE; (  calls intercept routine  )
  CASE0:=FALSE;   (  calls mathfunc seg.procedure )
  CASE0A:=FALSE;   (  calls mathfunc seg.procedure )
  CASE1:=FALSE;   (  calls rerun seg. procedure  )
  CASE2:=FALSE;   (  calls database seg.procedure )
  GRAFFLAG:=FALSE;(  flag for grafmode )
  RCTRFLAG:=FALSE;(  calls recenter routine  )
```

```
  QUITFLAG:=FALSE;(  calls tmafinit seg.procedure)
  STOPFLAG:=FALSE;(  terminates tma  )
END;

PROCEDURE INITFIG;
VAR I,J: INTEGER;
BEGIN
  FOR I:=1 TO 8 DO BEGIN
    FOR J:=1 TO 7 DO BEGIN
      SHIPFIG[I,J]:=FALSE;
    END;
  END;
  SHIPFIG[1,1]:= TRUE;
  SHIPFIG[4,1]:= TRUE;
  SHIPFIG[4,7]:= TRUE;
  SHIPFIG[7,3]:= TRUE;
  SHIPFIG[7,5]:= TRUE;
  FOR I:=1 TO 8 DO SHIPFIG[I,4]:= TRUE;
  FOR I:=2 TO 4 DO BEGIN
    SHIPFIG[I,3]:= TRUE;
    SHIPFIG[I,5]:= TRUE;
  END;
  FOR I:=3 TO 4 DO BEGIN
    SHIPFIG[I,2]:= TRUE;
    SHIPFIG[I,6]:= TRUE;
  END;
END;

BEGIN  ( TMAINIT )
  LEVEL0:=['0','1','2','3'];
  TIME:=0;
  XCTR:=140;
  YCTR:=96;
  MAXORANGE:=0;
  MAXBLUE:=0;
  BLUEMOVE:=0;
  ORANGEMOVE:=0;
  SCALE:=1;
  IF NOT DBCALLED THEN SETFLAGS;
  INITFIG;
  PRESTART;
  INITFLAG:=FALSE;
  IF OLDFLAG THEN EXIT(TMAINIT);
  START:
  CASE GFLAG OF
    '0' : CASE0:=TRUE;
    '1' : CASE1:=TRUE;
    '2' : CASE2:=TRUE;
  END;  ( CASE )
  IF GFLAG = '3' THEN STOPFLAG:=TRUE;
```

```
 END; ( TMAINIT )
SEGMENT PROCEDURE RESUME;
VAR
   BNOW,BOLD,ONOW,OOLD: FILE OF REAL;
   MISC: FILE OF INTEGER;
   J: INTEGER;

PROCEDURE MSG;
BEGIN
  GOTOXY(0,0); CRT(ERASEOS); GOTOXY(0,3);
  WRITELN('  UNABLE TO RESUME AN OLDGAME  --  DATA NOT AVAILABLE ');
  WRITELN;
  WRITELN('  MAKE SURE THE CORRECT DISK IS IN DRIVE #2 ! ');
  FOR J:=1 TO 4000 DO;
  INITFLAG:=TRUE;
  OLDFLAG:=FALSE;
  EXIT(RESUME);
END;

BEGIN
  J:=1;
  {$I-}
  RESET(BNOW, '#5:OLD1.DATA');
  IF (IORESULT <> 0) THEN MSG;
  {$I+}
  WHILE NOT EOF(BNOW) DO BEGIN
    BXY[1,J]:=BNOW^; GET(BNOW);
    BXY[2,J]:=BNOW^; GET(BNOW);
    GCUS[J]:= BNOW^; GET(BNOW);
    GSPD[J]:= BNOW^; GET(BNOW);
    J:=J+1;
  END;
  MAXBLUE:=J-1;
  CLOSE(BNOW,PURGE);
  J:=1;
  {$I-}
  RESET(BOLD, '#5:OLD2.DATA');
  IF (IORESULT <> 0) THEN MSG;
  {$I+}
  WHILE NOT EOF(BOLD) DO BEGIN
    OLDBXY[1,J]:=BOLD^; GET(BOLD);
    OLDBXY[2,J]:=BOLD^; GET(BOLD);
    J:=J+1;
  END;
  CLOSE(BOLD,PURGE);
  J:=1;
  {$I-}
  RESET(ONOW, '#5:OLD3.DATA');
  IF (IORESULT <> 0) THEN MSG;
  {$I+}
```

```
  WHILE NOT EOF(ONOW) DO BEGIN
    OXY[1,J]:=ONOW^; GET(ONOW);
    OXY[2,J]:=ONOW^; GET(ONOW);
    J:=J+1;
  END;
  MAXORANGE:=J-1;
  CLOSE(ONOW,PURGE);
  J:=1;
  {$I-}
  RESET(OOLD, '#5:OLD4.DATA');
  IF (IORESULT <> 0) THEN MSG;
  {$I+}
  WHILE NOT EOF(OOLD) DO BEGIN
    OLDOXY[1,J]:=OOLD^; GET(OOLD);
    OLDOXY[2,J]:=OOLD^; GET(OOLD);
    J:=J+1;
  END;
  CLOSE(OOLD,PURGE);
  {$I-}
  RESET(MISC, '#5:OLD5.DATA');
  IF (IORESULT <> 0) THEN MSG;
  {$I+}
  TIME:=MISC^; GET(MISC);
  BLUEMOVE:=MISC^; GET(MISC);
  ORANGEMOVE:=MISC^;
  CLOSE(MISC,PURGE);
  REWRITE(ORNHIST, '#4:ORNGOLD.DATA');
  BLFLAG:=TRUE;
  CONTINUE:=TRUE;
  OLDFLAG:=FALSE;
  OLDGAME:=TRUE;
  SCALE:=1;
  XCTR:=140;
  YCTR:=96;
  REWRITE(BLUHIST, '#4:BLUEOLD.DATA');
  INITTURTLE; TEXT:=FALSE;
END;

{$G+}
SEGMENT PROCEDURE DATABASE;
{   PROGRAM TO USE RANDOM ACCESS DISK FILES              }
{   AND TERMINAL-INDEPENDENT SCREEN CONTROL.             }
CONST
  CLASSLEN=10;NAMELEN=10;HULLNOLEN=8;SCOURSELEN=3;SSPOLEN=2;
  XPOSLEN=3;YPOSLEN=3;MSPDLEN=4;NMSLLEN=2;

FUNCTION YES: BOOLEAN;
BEGIN
  YES:= GETCHAR(['Y','y','N','n']) IN ['Y','y'];
END;
```

```
PROCEDURE MENU;
VAR SUCCESSFUL: BOOLEAN;
BEGIN
  GOTOXY(0,0); CRT(ERASEOS); GOTOXY(0,2);
  WRITELN('         TMA DATABASE MAIN MENU    ');
  WRITELN;
  WRITELN('     (A)  View  database ');
  WRITELN('     (B)  Change  database ');
  WRITELN('     (C)  Build new game database ');
  WRITELN('     (Q)  Quit  ');
  WRITELN;
  CH:=GETCHAR(['A','a','B','b','C','c','D','d','Q','q']);
  GOTOXY(0,0); CRT(ERASEOS);
  IF CH IN ['C','c'] THEN
    BUILDING:=TRUE
  ELSE
    BUILDING:=FALSE;
END;  (  Menu  )

PROCEDURE ZEROREC(VAR REC: SHIP);
BEGIN
  WITH REC DO
    BEGIN
      CLASS:='';
      NAME:='';
      HULLNO:='';
      SCOURSE:=0;
      SSPEED:=0;
      XPOS:=0;
      YPOS:=0;
      MAXSSPD:=0;
      NSAM:=0;
      SAMSPD:=0;
      NSSM:=0;
      SSMSPD:=0;
    END;
END;


PROCEDURE VALIDATE(VAR REC: SHIP);
{                                                    }
{                                                    }
{   TRIES TO DETECT AND ZERO AN UNINITIALIZED RECORD }
{   NO CHANGE IF ALL FIELDS ARE VALID.               }
{                                                    }
{                                                    }
  PROCEDURE CHECK(VAR S: STRING; MAXLEN: INTEGER);
  LABEL 1;
  VAR I: INTEGER;
```

```
  BEGIN
    IF LENGTH(S) > MAXLEN THEN GOTO 1;
    FOR I:=1 TO LENGTH(S) DO
      IF NOT (S[I] IN [' '..'}']) THEN GOTO 1;
    EXIT(CHECK); (  STRING IS OK  )
  1: ZEROREC(REC); EXIT(VALIDATE);
  END; (  CHECK  )

BEGIN (  VALIDATE  )
  WITH REC DO
    BEGIN
      CHECK(CLASS,CLASSLEN);
      CHECK(HULLNO,HULLNOLEN);
    END;
END; (  VALIDATE  )


PROCEDURE SHOWREC(REC: SHIP);
BEGIN
  GOTOXY(0,2); CRT(ERASEOS);
  WITH REC DO
    IF NOT BUILDING THEN
      BEGIN
        WRITELN('Class:         ',CLASS);
        WRITELN('Hull no.:      ',HULLNO);
        WRITELN('Max speed:     ',MAXSSPD);
        WRITELN('# SAM:         ',NSAM);
        WRITELN('SAM Speed:     ',SAMSPD);
        WRITELN('# SSM:         ',NSSM);
        WRITELN('SSM Speed:     ',SSMSPD);
      END
    ELSE
      BEGIN
        WRITELN('Class:         ',CLASS);
        WRITELN('Name:          ',NAME);
        WRITELN('Hull no.:      ',HULLNO);
        WRITELN('Course:        ',SCOURSE);
        WRITELN('Speed:         ',SSPEED);
        WRITELN('X Coordinate   ',XPOS);
        WRITELN('Y Coordinate   ',YPOS);
        WRITELN('Max speed:     ',MAXSSPD);
        WRITELN('# SAM:         ',NSAM);
        WRITELN('SAM Speed:     ',SAMSPD);
        WRITELN('# SSM:         ',NSSM);
        WRITELN('SSM Speed:     ',SSMSPD);
    END;
END;


PROCEDURE CHANGEREC(VAR REC: SHIP;SSPDLEN,MSPDLEN,NMSLLEN,NAMELEN,SCOURSELEN,
```

```
                    XPOSLEN,YPOSLEN : INTEGER);
  BEGIN
  IF BUILDING THEN
    BEGIN
    GOTOXY(0,15); CRT(ERASEOS);
    PROMPTAT(15,'(Press return for no change)');
    END
  ELSE
    BEGIN
    GOTOXY(0,12); CRT(ERASEOS);
    PROMPTAT(12,'(Press return for no change)')
  END;
  WITH REC DO
    IF NOT BUILDING THEN
    BEGIN
      GOTOXY(0,14);WRITELN('  ALL NUMERICAL INPUT ARE INTEGERS!! ');
      WRITE('Class:          '); GETSTRING(CLASS,CLASSLEN);  WRITELN;
      WRITE('Hull no.:       '); GETSTRING(HULLNO,HULLNOLEN); WRITELN;
      WRITE('Max speed:      '); GETINTEGER(MAXSSPD,SSPDLEN);WRITELN;
      WRITE('# SAM:          '); GETINTEGER(NSAM,NMSLLEN);WRITELN;
      WRITE('SAM Speed:      '); GETINTEGER(SAMSPD,MSPDLEN);WRITELN;
      WRITE('# SSM:          '); GETINTEGER(NSSM,NMSLLEN);WRITELN;
      WRITE('SSM Speed:      '); GETINTEGER(SSMSPD,MSPDLEN);WRITELN;
    END
    ELSE
    BEGIN
      GOTOXY(0,17);WRITELN('  ALL NUMERICAL INPUT ARE INTEGERS!! ');
      WRITE('Name:                  '); GETSTRING(NAME,NAMELEN); WRITELN;
      WRITE('Course:       (0-360) ');
      REPEAT
      GETINTEGER(SCOURSE,SCOURSELEN);
      UNTIL (SCOURSE <= 360) AND (SCOURSE >= 0);WRITELN;
      WRITE('Speed:      (0-Maxspd) ');
      REPEAT
      GETINTEGER(SSPEED,SSPDLEN);
      UNTIL (SSPEED >= 0) AND (SSPEED <= MAXSSPD );WRITELN;
      WRITE('X Coordinate: (0-100) ');
      REPEAT
      GETINTEGER(XPOS,XPOSLEN);
      UNTIL( XPOS >=0) AND (XPOS <=100);WRITELN;
      WRITE('Y Coordinate: (0-100) ');
      REPEAT
      GETINTEGER(YPOS,YPOSLEN);
      UNTIL(YPOS >=0) AND (YPOS <= 100);WRITELN;
    END;
END; (  CHANGEREC  )


PROCEDURE NEWFILE;
```

```
VAR SUCESSFUL: BOOLEAN;
    IREC,MAXREC: INTEGER;

BEGIN
  BUILDING:=FALSE;
  CLOSE(DATAFILE,LOCK); (  IN CASE IT'S ALREADY OPEN  )
  (*$I-*)
  REPEAT
    GOTOXY(0,1); CRT(ERASEOS);
    GOTOXY(0,2);
    WRITELN('     TMA Master datafile is : BLUEMASTER ');
    WRITELN('     TMA Game datafile is   : BLUEGAME ');
    WRITELN('     For new database use different name. ');
    PROMPTAT(8,'File Name: '); READLN(FILENAME);
    RESET(DATAFILE,FILENAME);  (  TRY TO OPEN AN OLD FILE  )
    SUCESSFUL := (IORESULT=0);
    IF NOT SUCESSFUL THEN  (  START A NEW FILE?  )
      BEGIN
        PROMPTAT(10,'Start a new file ? ');
        IF YES THEN
          BEGIN
            REWRITE(DATAFILE,FILENAME);
            PROMPTAT(12,'Reserve how many records ? ');
            READLN(MAXREC);
            SEEK(DATAFILE,MAXREC);
            ZEROREC(DATAFILE^);
            (*$I-*)
            PUT(DATAFILE);
            (*$I+*)
            IF (IORESULT<>0) OR EOF(DATAFILE) THEN
              BEGIN
                PROMPTAT(14,'Not enough room. Press return ');
                READLN;
                SUCESSFUL:=FALSE;
              END
            ELSE
              BEGIN
                 ( INITIALIZE CONTENTS OF FILE )
                FOR IREC:=0 TO MAXREC DO
                  BEGIN
                    SEEK(DATAFILE,IREC);
                    PUT(DATAFILE);
                  END;
                CLOSE(DATAFILE,LOCK); (  LOCK IT IN PLACE  )
                RESET(DATAFILE,FILENAME);
                SUCESSFUL := (IORESULT = 0);
              END;
          END;
      END;
  UNTIL SUCESSFUL;
```

```
  (*$I+*)
  RECNUM:=-1;
  LASTCHANGE:=FALSE;
  MENU;
END;


PROCEDURE CHANGE;
BEGIN
  (*$I-*)
  REPEAT
    PROMPTAT(2,'Change which record ? ');
    READLN(RECNUM);
  UNTIL IORESULT=0;
  (*$I+*)
  SEEK(DATAFILE,RECNUM);
  GET(DATAFILE);
  IF EOF(DATAFILE) THEN (  EXTENDING FILE  ) ZEROREC(DATAFILE^);
  VALIDATE(DATAFILE^);
  SHOWREC(DATAFILE^);
  CHANGEREC(DATAFILE^,SSPDLEN,MSPDLEN,NMSLLEN,NAMELEN,SCOURSELEN,
            XPOSLEN,YPOSLEN);
  SEEK(DATAFILE,RECNUM);
  (*$I-*)
  PUT(DATAFILE);
  (*$I+*)
  IF (IORESULT<>0) OR EOF(DATAFILE) THEN
    BEGIN
      GOTOXY(0,20);
      WRITELN(CHR(7),'UNABLE TO EXTEND FILE, NO DATA WRITTEN');
      WRITELN('Use Filer K(runch command to make space after file.');
    END;
  LASTCHANGE:=TRUE;
END;


PROCEDURE VIEW;
BEGIN
  (*$I-*)
  REPEAT
    PROMPTAT(2,'View which record ? ');
    READLN(RECNUM);
  UNTIL IORESULT=0;
  (*$I+*)
  SEEK(DATAFILE,RECNUM);
  GET(DATAFILE);
  IF EOF(DATAFILE) THEN
    BEGIN
      GOTOXY(0,4);
      WRITE('Record ',RECNUM,' not in file.');
```

```
    END
  ELSE
    BEGIN
      VALIDATE(DATAFILE^);
      SHOWREC(DATAFILE^);
    END;
  LASTCHANGE:=FALSE;
END;


PROCEDURE NEXT;
 ( VIEW OR CHANGE NEXT RECORD  )
BEGIN
  RECNUM:=RECNUM+1;
  SEEK(DATAFILE,RECNUM);
  GET(DATAFILE);
  IF EOF(DATAFILE) THEN
    BEGIN
      ZEROREC(DATAFILE^);
      IF NOT LASTCHANGE THEN
        BEGIN
          GOTOXY(0,4);
          WRITE('Record ',RECNUM,' not in file.');
          EXIT(NEXT);
        END;
    END;
  GOTOXY(0,1); WRITE('Record number ',RECNUM);
  VALIDATE(DATAFILE^);
  SHOWREC(DATAFILE^);
  IF LASTCHANGE THEN
    BEGIN
      CHANGEREC(DATAFILE^,SSPOLEN,MSPOLEN,NMSLLEN,NAMELEN,SCOURSELEN,
                XPOSLEN,YPOSLEN);
      SEEK(DATAFILE,RECNUM);
      {$I-}
      PUT(DATAFILE);
      {$I+}
      IF (IORESULT<>0) OR EOF(DATAFILE) THEN
        BEGIN
          GOTOXY(0,20);
          WRITELN(CHR(7),'UNABLE TO EXTEND FILE, NO DATA WRITTEN');
          WRITELN('Use Filer K(runch command to make space after file.');
        END;
    END;
END;

PROCEDURE INSTRUCTIONS;

VAR
  CH : CHAR;
```

```
BEGIN
  WRITELN('         MASTER TMA DATABASE EDITOR ');
  WRITELN;
  WRITELN('      This module of the program allows you (the user) to ');
  WRITELN('    VIEW or CHANGE an existing data base. ');
  WRITELN('      In addition you may create a new data base by using the ');
  WRITELN('    FILE command.');
  WRITELN;
  WRITELN('      You begin by entering the FILE NAME of the data your ');
  WRITELN('    interested in or the FILE NAME of your NEW data base. ');
  WRITELN;
  WRITELN('      For example:  the file ORANGEMASTER contains the present ');
  WRITELN('    library of Orange ships in the Master TMA data base. ');
  WRITELN('    Likewise, for the file BLUEMASTER. ');
  WRITELN;
  WRITELN;
  WRITELN;
  WRITELN;
  WRITELN('     (  HIT SPACEBAR TO CONTINUE ) ');
  READ(KEYBOARD,CH);
END;   ( INSTRUCTIONS )

PROCEDURE HELP;
BEGIN
  GOTOXY(0,4); CRT(ERASEOS);
  WRITELN(' * HELP    :    Explanation of options * ');
  WRITELN;
  WRITELN('   V)iew   : To look at the selected database ');
  WRITELN('   C)hange : To change any/all elements of the selected database');
  WRITELN('   N)ext   : To page forward to the next record ');
  WRITELN('   F)ile   : To close the file being worked on and start another');
  WRITELN('   R)eturn : To return to the master menu ');
  WRITELN('   H)elp   : To display this table of explanations ');
  WRITELN('   Q)uit   : To terminate the database manipulation module ')
END;


BEGIN (  MAIN PROGRAM  )
  CLOSE(BLUGAME,LOCK);
  CLOSE(BLUHIST,LOCK);
  CLOSE(ORNGAME,LOCK);
  CLOSE(ORNHIST,LOCK);
  CLOSE(ATIME,LOCK);
  CASE2:= FALSE;
  INITFLAG:=TRUE;
  GOTOXY(0,0); CRT(ERASEOS);
  INSTRUCTIONS;
  NEWFILE;
  DBCALLED:=TRUE;
```

```
  REPEAT
    IF NOT BUILDING THEN
PROMPTAT(0,'DBMASTER: V(iew, C(hange, N(ext, F(ile, R(eturn, H)elp, Q)uit')
    ELSE
PROMPTAT(0,'DBBUILDER: V(iew, C(hange, N(ext, F(ile, R(eturn, H)elp, Q(uit');
    CH:=GETCHAR(['N','n','F','f','V','v','C','c','R','r','H','h','Q','q']);
    CRT(ERASEOS);
    CASE CH OF
      'N','n': NEXT;
      'F','f': NEWFILE;
      'V','v': VIEW;
      'C','c': CHANGE;
      'R','r': MENU;
      'H','h': HELP;
    END;
  UNTIL CH IN ['Q','q'];
  CLOSE(DATAFILE,LOCK);
  PROMPTAT(12,'FINISH DATA BASE MANIPULATION...');
END;


SEGMENT PROCEDURE MATHFUNC;

VAR
   INDEX: INTEGER;
   L: CHAR;
   FNAME: STRING[15];
   GNAME: STRING[10];
   GCOURSE,GSPEED,GX,GY: REAL;
   SUCCESSFUL: BOOLEAN;

PROCEDURE MSTART;
VAR
  SUCCESSFUL: BOOLEAN;
BEGIN
  SUCCESSFUL:=TRUE;
  IF DBCALLED THEN FNAME:=FILENAME
  ELSE
    REPEAT
      DBCALLED:=TRUE;
      GOTOXY(0,0);CRT(ERASEOS);
      GOTOXY(0,2);
      IF NOT SUCCESSFUL THEN
      WRITELN('   That File Name is not on this disk!! ');
      WRITELN;
      WRITELN('   MEMORY AVAILABLE IN WORDS: ',MEMAVAIL);
      WRITELN;WRITELN;
      WRITELN('   Default TMA game file name is BLUEGAME ');
      WRITELN('   If you have previously saved another game ');
      WRITELN('   database you wish to use type that file name in. ');
```

```
      PROMPTAT(12,'  File Name: ');READLN(FNAME);
      FILENAME:=FNAME;
      {$I-}
      RESET(DATAFILE,FNAME);  (TRY TO OPEN AN OLD FILE)
      SUCCESSFUL:=(IORESULT=0);
      {$I+}
    UNTIL SUCCESSFUL;
    CLOSE(DATAFILE,LOCK);
END;

PROCEDURE MINIT(VAR REC: SHIP);

PROCEDURE TRANSFORM;
VAR
  TWOPI: REAL;
BEGIN  ( TRANSFORM )
 TWOPI:=2.0*PI;
 GX:= GX*2.8;
 GY:= GY*1.92;
 GCOURSE:=GCOURSE*(PI/180.0);
 WHILE GCOURSE > TWOPI DO
   BEGIN
     GCOURSE:= (GCOURSE - TWOPI);
   END;
END;  ( TRANSFORM )

PROCEDURE INITDISP;
VAR  X,Y: INTEGER;
BEGIN  ( INITDISP )
  PENCOLOR (NONE);
  X:=ROUND(GX);
  Y:=ROUND(GY);
  MOVETO(X,Y);
  DRAWBLOCK(SHIPFIG,2,0,0,7,3,X,Y,10);
  NUMBER(X,Y,INDEX);
END;   ( INITDISP )

PROCEDURE SETARRAY;
BEGIN  (  SETARRAY  )
  GCUS[INDEX]:=GCOURSE;
  GSPD[INDEX]:=GSPEED;
  BXY[1,INDEX]:=GX;
  BXY[2,INDEX]:=GY;
  BLUGAME^:=GX;PUT(BLUGAME);
  BLUGAME^:=GY;PUT(BLUGAME);
  BLUHIST^:=GX;PUT(BLUHIST);
  BLUHIST^:=GY;PUT(BLUHIST);
END;  (  SETARRAY  )

BEGIN  (  MINIT  )
```

```
  INITTURTLE; TEXTMODE; TEXT:= TRUE;
  RECNUM:=0;
  RESET(DATAFILE,FNAME);
  REWRITE(BLUHIST, '#4:BLUEOLD.DATA');
  REWRITE(BLUGAME, '#5:BLUENOW.DATA');
  WHILE (NOT EOF(DATAFILE)) AND (RECNUM < 10) DO BEGIN
    SEEK(DATAFILE,RECNUM);
    GET(DATAFILE);
    WITH REC DO
      IF (LENGTH(CLASS) > 1) AND (MAXSSPD > 0) THEN BEGIN
        INDEX:=RECNUM+1;
        MAXBLUE:=INDEX;
        GNAME:=NAME;
        GCOURSE:=SCOURSE;
        GSPEED:=SSPEED;
        GX:=XPOS;
        GY:=YPOS;
        TRANSFORM;
        SETARRAY;
        INITDISP;
      END;
      RECNUM:=RECNUM+1;
    END;
    CLOSE(DATAFILE,LOCK);
    CLOSE(BLUGAME,LOCK);
    BLUHIST^:= -9999; PUT(BLUHIST);
    GRAFMODE; TEXT:=FALSE;
    BLUEMOVE:=BLUEMOVE + 1;
    CASE0A:=TRUE;
    BLFLAG:=TRUE;
    EXIT(MATHFUNC);
END;  {  MINIT  }

PROCEDURE GETDATA;
VAR J: INTEGER;
BEGIN
  REWRITE(ORNHIST, '#4:ORNGOLD.DATA');
  J:=1;
  {$I-}
  RESET(ORNGAME, '#5:ORNGNOW.DATA');
  SUCCESSFUL:= (IORESULT=0);
  IF NOT SUCCESSFUL THEN EXIT(GETDATA);
  {$I+}
  WHILE (NOT EOF(ORNGAME)) AND (J <= 10) DO BEGIN
    OXY[1,J]:=ORNGAME^;
    GET(ORNGAME);
    OXY[2,J]:=ORNGAME^;
    ORNHIST^:=OXY[1,J]; PUT(ORNHIST);
    ORNHIST^:=OXY[2,J]; PUT(ORNHIST);
    J:=J + 1;
```

```
    GET(ORNGAME);
  END;
  MAXORANGE:= J - 1;
  ORNHIST^:= -9999; PUT(ORNHIST);
  CLOSE(ORNGAME,LOCK);
END;

PROCEDURE FIRSTMOVE;
BEGIN  ( firstmove )
  CONTINUE:=TRUE;
  TEXTMODE; TEXT:=TRUE;
  SUCCESSFUL:=TRUE;
  REWRITE(ATIME, '#4:TIME.DATA');
  ATIME^:=TIME; PUT(ATIME);
  REPEAT
    GOTOXY(0,0); CRT(ERASEOS);
    GOTOXY(0,3);
    WRITELN(' !!!  FOLLOW THESE INSTRUCTIONS  !!!  ');
    WRITELN;
    WRITELN;
    IF NOT SUCCESSFUL THEN
    WRITELN(' * ERROR HAS OCCURED WHILE TRYING TO READ DATAFILE ON DISK #2 *')
    WRITELN;
    WRITELN(' 1.  REMOVE GAME DISK FROM DRIVE # 2 ');
    WRITELN;
    WRITELN(' 2.  EXCHANGE DISKS WITH OPPONENT  ');
    WRITELN;
    WRITELN(' 3.  PLACE OPPONENT'S UPDATED GAME DISK IN DRIVE # 2 ');
    WRITELN; WRITELN;
    WRITELN('         (  PRESS RETURN WHEN READY  )  ');
    WRITELN;READLN;
    GETDATA;
  UNTIL SUCCESSFUL;
  ORANGEMOVE:=ORANGEMOVE + 1;
  BLFLAG:=TRUE;
END;

PROCEDURE MENUM;
BEGIN
    TEXTMODE; TEXT:= TRUE;
    OKSET:=['A','a','B','b','Q','q'];
    GOTOXY(0,0):CRT(ERASEOS);
    PROMPTAT(0,') You have the following choices ');
    GOTOXY(0,3);
    WRITELN('  A) Return to DATABASE ( for changes ) ');
    WRITELN('  B) Continue with the game');
    WRITELN;
    WRITELN('  Q)uit (terminates program) ');
    L:=GETCHAR(OKSET);
    CASE L OF
```

```
      'A','a': CASE2:=TRUE;
      'B','b': FIRSTMOVE;
      'Q','q': EXIT(TMA);
    END;
END;

BEGIN  ( MATHFUNC )
  (*$R TURTLEGRAPHICS*)
  (*$R THESIS0*)
  CASE0:=FALSE;
  TIME:=0;
  IF NOT CASE0A THEN MSTART;
  IF NOT CASE0A THEN MINIT(DATAFILE^);
  CASE0A:=FALSE;
  REPEAT
    OKSET:=[' '];
    L:=GETCHAR(OKSET + [CHR(27),CHR(8),CHR(21)]);
    IF L=CHR( 8) THEN L:='D';
    IF L=CHR(21) THEN L:='U';
    CASE L OF
       'U' : BEGIN
               UPSCALE;
               BLFLAG:=TRUE;
               CASE0A:=TRUE;
               EXIT(MATHFUNC);
             END;
       'D' : BEGIN
               DOWNSCALE;
               BLFLAG:=TRUE;
               CASE0A:=TRUE;
               EXIT(MATHFUNC);
             END;
       ' ' : BEGIN
               RCTRFLAG:=TRUE;
               CASE0A:=TRUE;
               EXIT(MATHFUNC);
             END;
    END;
  UNTIL L = CHR(27);
  MENUM;
END;
SEGMENT PROCEDURE MOTION;

VAR
   MFLAG1: BOOLEAN;

PROCEDURE DISTSPD(DX,DY: REAL;VAR DIST,SPD: REAL);
BEGIN
  DIST:= SQRT(SQR(DX) + SQR(DY));
  SPD:= DIST / DELTAT * 60;
end;
```

```
FUNCTION DEGREES(DX,DY: REAL) : INTEGER;
CONST  ERR = 0.01;
VAR VALUE: INTEGER;
BEGIN
  IF (DX < (0+ERR)) AND (DX > (0-ERR)) THEN BEGIN
    IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN DEGREES := 0;
    IF DY > 0 THEN DEGREES := 180;
    IF DY < 0 THEN DEGREES := 360;
    EXIT(DEGREES);
  END;
  IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN BEGIN
    IF DX > 0 THEN DEGREES := 270;
    IF DX < 0 THEN DEGREES := 090;
    EXIT(DEGREES);
  END;
  VALUE := ROUND(180 / PI * ATAN (DX/DY));
  IF ABS (VALUE) <> VALUE THEN BEGIN
    IF DX < 0 THEN DEGREES:= 180 + VALUE;
    IF DX > 0 THEN DEGREES:= 360 + VALUE;
  END
  ELSE BEGIN
    IF DX > 0 THEN DEGREES:= 180 + VALUE;
    IF DX < 0 THEN DEGREES:= VALUE;
  END;
END;

PROCEDURE DISPLAY;
VAR J,X,Y: INTEGER;
BEGIN
  PENCOLOR(NONE);
  FOR J:= 1 TO MAXBLUE DO BEGIN
    X:=ROUND((BXY[1,J]-XCTR)/SCALE)+140;
    Y:=ROUND((BXY[2,J]-YCTR)/SCALE)+96;
    MOVETO(X,Y);
    DRAWBLOCK(SHIPFIG,2,0,0,7,3,X,Y,10);
    NUMBER(X,Y,J);
  END;
  FOR J:=1 TO MAXORANGE DO BEGIN
    X:=ROUND((OXY[1,J]-XCTR)/SCALE)+140;
    Y:=ROUND((OXY[2,J]-YCTR)/SCALE)+96;
    MOVETO(X,Y);
    DRAWBLOCK(SHIPFIG,2,0,0,7,3,X,Y,5);
    LETTER(X,Y,J);
  END;
  GRAFMODE;TEXT:=FALSE;
END;

PROCEDURE YOURSIDE(TGT: CHAR;VAR X,Y: INTEGER);
VAR J: INTEGER;
```

```
    XX,YY: REAL;
BEGIN
  J:=ORD(TGT)-ORD('0');
  J:=J+1;
  X:=ROUND(BXY[1,J]);


  Y:=ROUND(BXY[2,J]);
END;

PROCEDURE OTHERSIDE(TGT: CHAR;VAR X,Y: INTEGER);
VAR J: INTEGER:
    XX,YY: REAL;
BEGIN
IF TGT IN ['a'..'j'] THEN J:=ORD(TGT)-ORD('a') ELSE
  J:=ORD(TGT)-ORD('A');
  J:=J+1;
  X:=ROUND(OXY[1,J]);
  Y:=ROUND(OXY[2,J]);
END;

PROCEDURE RANGE;
VAR
   X1,X2,Y1,Y2,DIST,X,Y,I: INTEGER;
   DISTANCE,DELTAX,DELTAY: REAL;
   TARGET1,TARGET2: CHAR;
BEGIN
  X:=0;Y:=0;
  GRAFMODE; TEXT:=FALSE;
  OKSET:=['0'..'9','A'..'J','a'..'j'];
  PENCOLOR(WHITE);
  BOXSTRING(3,2,'  RANGE:  LETTER / NUMBER OF 1ST SHIP  ');
  TARGET1:=GETCHAR(OKSET);
  IF TARGET1 IN['0'..'9'ITHEN YOURSIDE(TARGET1,X,Y)ELSE OTHERSIDE(TARGET1,X,Y)
  X1:=X; Y1:=Y;
  BOXSTRING(3,2,'  RANGE:  LETTER / NUMBER OF 2ND SHIP  ');
  TARGET2:=GETCHAR(OKSET);
  CLEARBOX;
  IF TARGET2 IN['0'..'9'ITHEN YOURSIDE(TARGET2,X,Y)
    ELSE OTHERSIDE(TARGET2,X,Y);
  X2:=X; Y2:=Y;
  DELTAX:= X2 - X1; DELTAY:= Y2 - Y1;
  DISTANCE:=SQRT(SQR(DELTAX / 2.8) + SQR(DELTAY / 1.92));
  DIST:=ROUND(DISTANCE);
  X1:=ROUND((X1-XCTR)/SCALE)+140;
  Y1:=ROUND((Y1-YCTR)/SCALE)+96;
  MOVETO(X1,Y1);
  PENCOLOR(WHITE);
  X2:=ROUND((X2-XCTR)/SCALE)+140;
  Y2:=ROUND((Y2-YCTR)/SCALE)+96;
  MOVETO(X2,Y2);
  PENCOLOR(NONE);
```

```
  FOR I:= 1 TO 2000 DO BEGIN
    I:=I+1;
  END;
  GOTOXY(0,0);CRT(ERASEOS);
  GOTOXY(0,5);
  WRITELN('    The distance from ',TARGET1,' to ',TARGET2,' is : ');
  WRITELN;
  WRITELN('              ',DIST,' MILES! ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('           ( PRESS RETURN TO CONTINUE )   ');
  TEXTMODE;TEXT:=TRUE;
  READLN;
END;

PROCEDURE INTERROGATE;
VAR COURSE,X,Y,TGT:INTEGER;
    TARGET: CHAR;
    SPD.DX,DY,DIST: REAL;

PROCEDURE MESSAGE2;
BEGIN
  GOTOXY(0,0); CRT(ERASEOS);
  GOTOXY(0,5);
  WRITELN('    The course and speed of ship # ',TARGET,' is : ');
  WRITELN;
  WRITELN('     ',COURSE,' Degrees  ',ROUND(SPD),' Knots ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('         ( PRESS RETURN TO CONTINUE )   ');
  TEXTMODE;TEXT:=TRUE;
  READLN;
END;

PROCEDURE MESSAGE1;
BEGIN
  TEXTMODE; TEXT:= TRUE;
  GOTOXY(0,0);CRT(ERASEOS);
  GOTOXY(0,5);
  WRITELN('  Not enough data to comoute course / speed ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('      ( PRESS RETURN TO CONTINUE )   ');
  READLN;
  EXIT(INTERROGATE);
END;

BEGIN (  interrogate  )
  IF ORANGEMOVE <= 1 THEN MESSAGE1;
  X:=0; Y:=0;
  OKSET:=['0'..'9','A'..'J','a'..'j'];
  GRAFMODE; TEXT:=FALSE;
  BOXSTRING(3,2,' INTERR: ENTER LETTER / NUMBER OF SHIP ');
```

```
  TARGET:=GETCHAR(OKSET);
  CLEARBOX;
  IF TARGET IN ['0'..'9']THEN BEGIN
    TGT:=ORD(TARGET) - ORD('0');
    TGT:=TGT + 1;
    DX:=OLDBXY[1,TGT] - BXY[1,TGT];
    DY:=OLDBXY[2,TGT] - BXY[2,TGT];
    END
  ELSE BEGIN
    IF TARGET IN ['a'..'j'] THEN TGT:=ORD(TARGET)-ORD('a') ELSE
      TGT:=ORD(TARGET) - ORD('A');
    TGT:=TGT + 1;
    DX:=OLDOXY[1,TGT] - OXY[1,TGT];
    DY:=OLDOXY[2,TGT] - OXY[2,TGT];
  END;
  DX:=DX / 2.8;
  DY:=DY / 1.92;
  TEXTMODE; TEXT:=TRUE;
  COURSE:=DEGREES(DX,DY);
  DISTSPD(DX,DY,DIST,SPD);
  MESSAGE2;
END;

PROCEDURE STARTIT;
BEGIN
  IF GRAFFLAG THEN BEGIN
    GRAFMODE;
    TEXT:=FALSE;
    GRAFFLAG:=FALSE;
  END
  ELSE BEGIN
    TEXTMODE;
    TEXT:=TRUE;
  END;
END;

PROCEDURE STOPIT;
BEGIN
  RRFLAG:=FALSE;
  TEXTMODE; TEXT:=TRUE;
  QUITFLAG:=TRUE;
  EXIT(MOTION);
END;

PROCEDURE MENU;
VAR
  CH: CHAR;
BEGIN
  STARTIT;
  REPEAT
```

```
OKSET:=['A'..'H','a'..'h','Q','q'];
GOTOXY(0,0);CRT(ERASEOS);
GOTOXY(0,5);
WRITELN(' A)  Advance to next time-step ');
WRITELN(' B)  Draw range circle around a specified unit ');
WRITELN(' C)  Compute range between units  ');
WRITELN(' D)  Determine CPA ( CLOSEST POINT OF APPROACH BETWEEN UNITS )');
WRITELN(' E)  Compute and display Course and Speed of units  ');
WRITELN(' F)  Check / Change movements of a unit  ');
WRITELN(' G)  Compute Intercept from one unit to another ');
WRITELN(' H)  Clear screen ');
WRITELN;
WRITELN(' Q)  QUIT THE TMA PROGRAM  ');
WRITELN;
WRITELN(' [<-]   Downscale display ');
WRITELN(' [->]   Upscale display ');
WRITELN('[space] Recenters display ');
WRITELN('[ESC]   Toggles display between text and graphics ');
CH:=GETCHAR(OKSET+ [CHR(27),CHR(8),CHR(21),CHR(32)]);
CRT(ERASEOS);
IF CH=CHR(8) THEN CH:='R';  ( left arrow ) reduce scale )
IF CH=CHR(21) THEN CH:='I'; ( right arrow ) increase scale )
IF (CH=CHR(27)) AND (TEXT) THEN BEGIN
  GRAFMODE;
  TEXT:=FALSE;
  CH:='Z';
END
ELSE IF (CH=CHR(27)) AND (NOT TEXT) THEN BEGIN
  TEXTMODE;
  TEXT:=TRUE;
  CH:='Z';
END;
CASE CH OF
   'A','a':  BEGIN
               TFLAG5:=TRUE;
               EXIT(MOTION);
             END;
   'B','b':  BEGIN
               TFLAG7:=TRUE;
               EXIT(MOTION);
             END:
   'C','c':  RANGE;
   'D','d':  BEGIN
               TFLAG3:=TRUE;
               EXIT(MOTION);
             END;
   'E','e':  INTERROGATE;
   'F','f':  BEGIN
               TFLAG1:=TRUE;
               EXIT(MOTION);
```

```
                END;
      'G','g':  BEGIN
                  ICPTFLAG:=TRUE;
                  BLFLAG:=TRUE;
                  TFLAG2:=TRUE;
                  EXIT(MOTION);
                END;
      'H','h':  BEGIN
                  INITTURTLE;
                  BLFLAG:=TRUE;
                  CONTINUE:=TRUE;
                  EXIT(MOTION);
                END;
      'R'    :  BEGIN
                  DOWNSCALE;
                  BLFLAG:=TRUE;
                  TFLAG2:=TRUE;
                  EXIT(MOTION);
                END;
      'I'    :  BEGIN
                  UPSCALE;
                  BLFLAG:=TRUE;
                  TFLAG2:=TRUE;
                  EXIT(MOTION);
                END;
      ' '    :  BEGIN
                  RCTRFLAG:=TRUE;
                  TFLAG2:=TRUE;
                  EXIT(MOTION);
                END;
    END;
    IF (CH = 'R') OR (CH = 'I') OR (CH =' ') THEN GRAFMODE;
  UNTIL CH IN [ 'Q','q' ];
  STOPIT;
END;

BEGIN ( motion )
(*$R TURTLEGRAPHICS*)
(*$R THESIS0*)
(*$R TRANSCEND*)
  IF (TFLAG6) OR (CONTINUE) THEN DISPLAY;
  CONTINUE:=FALSE;
  TFLAG2:=FALSE;
  TFLAG4:=FALSE;
  TFLAG6:=FALSE;
  MENU;
END;

SEGMENT PROCEDURE CIRCLE;
```

```
PROCEDURE CIRCPART;
VAR
   X,Y,NX,NY,I,RANGE,RANGELEN: INTEGER;
   RX,RY,RADIUS,THETA: REAL;
   TARGET: CHAR;
   DOT: BOOLEAN;

PROCEDURE YOURSIDE(TGT: CHAR;VAR X,Y: INTEGER);
VAR J: INTEGER;
BEGIN
  J:=ORD(TGT)-ORD('0');
  J:=J+1;
  X:=ROUND((BXY[1,J]-XCTR)/SCALE)+140;
  Y:=ROUND((BXY[2,J]-YCTR)/SCALE)+96;
END;

PROCEDURE OTHERSIDE(TGT: CHAR;VAR X,Y: INTEGER);
VAR J: INTEGER;
BEGIN
  IF TGT IN ['a'..'j'] THEN J:=ORD(TGT)-ORD('a') ELSE
  J:=ORD(TGT)-ORD('A');
  J:=J+1;
  X:=ROUND((OXY[1,J]-XCTR)/SCALE)+140:
  Y:=ROUND((OXY[2,J]-YCTR)/SCALE)+96:
END;

BEGIN ( circlepart )
(*$R TURTLEGRAPHICS*)
(*$R TRANSCEND*)
(*$R THESIS0*)
  TFLAG7:=FALSE;
  TFLAG2:=TRUE;
  DOT:=TRUE;
  RANGELEN:=3;
  GRAFMODE; TEXT:=FALSE;
  OKSET:=['0'..'9','A'..'J','a'..'j'];
  PENCOLOR(WHITE);
  BOXSTRING(3,2,' CIRCLE: ENTER LETTER/NUMBER OF ORIGIN ');
  TARGET:=GETCHAR(OKSET);
  IF TARGET IN ['0'..'9'] THEN YOURSIDE(TARGET,X,Y)
    ELSE OTHERSIDE(TARGET,X,Y);
  BOXSTRING(3,2,' CIRCLE: ENTER DESIRED RANGE OF CIRCLE ');
  GETINTEGER(RANGE,RANGELEN);
  CLEARBOX;
  RADIUS:=RANGE;
  MOVETO(X,Y);
  FOR I:= 1 TO 100 DO BEGIN
    THETA:=0.02*I*PI;
    RX:=(RADIUS*2.3*SIN(THETA)/SCALE+X);
    RY:=(RADIUS*1.92*COS(THETA)/SCALE+Y);
```

```
    NX:=ROUND (RX);
    NY:=ROUND (RY);
    DRAWBLOCK(DOT,1,0,0,1,1,NX,NY,10);
    MOVETO(X,Y);
  END;
END;

PROCEDURE ADV1PART;
VAR TESTVALUE: REAL;

PROCEDURE MOVEIT;

PROCEDURE MSG0;
BEGIN
  GOTOXY(0,0); CRT(ERASEOS); GOTOXY(0,3);
  WRITELN('  < THE RERUN PORTION OF TMA IS TERMINATED > ');
  WRITELN;
  WRITELN('       NO MORE DATA EXISTS ON FILE  ');
  WRITELN;
  WRITELN('    THANK-YOU FOR PLAYING TMA   --   PRESS RETURN TO CONTINUE  ');
  TEXTMODE; TEXT:=TRUE;
  READLN;
  TFLAG5:=FALSE;
  RRFLAG:=FALSE;
  QUITFLAG:=TRUE;
  EXIT(CIRCLE);
END;

PROCEDURE GETBLUE;
VAR J: INTEGER;
BEGIN
  J:=1;
  TESTVALUE:=0;
  WHILE (NOT EOF(BLUHIST)) AND (TESTVALUE > -999) DO BEGIN
    GET(BLUHIST);
    IF EOF(BLUHIST) THEN MSG0;
    IF J > 10 THEN EXIT(GETBLUE);
    IF TESTVALUE < -999 THEN EXIT(GETBLUE);
    OLDBXY[1,J]:= BXY[1,J];
    BXY[1,J]:=BLUHIST^;
    GET(BLUHIST);
    OLDBXY[2,J]:= BXY[2,J];
    BXY[2,J]:=BLUHIST^;
    J:=J + 1;
    MAXBLUE:= J - 1;
  END;
END;

PROCEDURE GETORANGE;
VAR J: INTEGER;
```

```
BEGIN
  J:=1;
  TESTVALUE:=0;
  WHILE (NOT EOF(ORNHIST)) AND (TESTVALUE > -999) DO BEGIN
    GET(ORNHIST);
    IF EOF(ORNHIST) THEN MSG0;
    IF J > 10 THEN EXIT(GETORANGE);
    IF TESTVALUE < -999 THEN EXIT(GETORANGE);
    OLDOXY[1,J]:= OXY[1,J];
    OXY[1,J]:=ORNHIST^;
    GET(ORNHIST);
    OLDOXY[2,J]:= OXY[2,J];
    OXY[2,J]:=ORNHIST^;
    J:=J + 1;
    MAXORANGE:= J - 1;
  END;
END;

BEGIN  { moveit }
  GETBLUE;
  BLUEMOVE:=BLUEMOVE + 1;
  GETORANGE;
  ORANGEMOVE:=ORANGEMOVE + 1;
  TFLAG6:=TRUE;
  GRAFFLAG:=TRUE;
  GET(ATIME);
  DELTAT:=ATIME^;
  TIME:=TIME + DELTAT;
  BLFLAG:=TRUE;
  EXIT(ADV1PART);
END;

PROCEDURE MENU;
VAR
  CH: CHAR;
BEGIN
  IF GRAFFLAG THEN BEGIN
    GRAFMODE;
    TEXT:= FALSE;
  END
  ELSE BEGIN
    TEXTMODE;
    TEXT:=TRUE;
  END;
  REPEAT
    OKSET:=['A','a','Q','q'];
    GOTOXY(0,0);CRT(ERASEOS);
    GOTOXY(0,5);
    WRITELN('  A)  Advance:  move ships to new positions based on time-step');
    WRITELN('  Q)  Quit   :  return to program's outer level ');
```

```
    WRITELN;WRITELN;WRITELN;
    WRITELN(' ***(  USE "ESC" KEY TO TOGGLE BETWEEN TEXT AND GRAPHICS )*** ');
    CH:=GETCHAR(OKSET+ [CHR(27),CHR(8),CHR(21),CHR(32)]);
    CRT(ERASEOS);
    IF CH=CHR(8) THEN CH:='R';  ( left arrow ) reduce scale )
    IF CH=CHR(21) THEN CH:='I'; ( right arrow ) increase scale )
    IF (CH=CHR(27)) AND (TEXT)·THEN BEGIN
      GRAPHMODE; TEXT:=FALSE;
      CH:='Z';
    END;
    IF (CH=CHR(27)) AND (NOT TEXT) THEN BEGIN
      TEXTMODE; TEXT:=TRUE;
      CH:='Z';
    END;
    CASE CH OF
       'A','a':  MOVEIT;
       'R'    :  BEGIN
                   DOWNSCALE;
                   BLFLAG:=TRUE;
                   TFLAGS:=TRUE;
                   EXIT(ADV1PART);
                 END;
       'I'    :  BEGIN
                   UPSCALE;
                   BLFLAG:=TRUE;
                   TFLAGS:=TRUE;
                   EXIT(ADV1PART);
                 END;
       ' '    :  BEGIN
                   RCTRFLAG:=TRUE;
                   TFLAGS:=TRUE;
                   EXIT(ADV1PART);
                 END;
    END;
    IF (CH = 'R') OR (CH = 'I') OR (CH =' ') THEN BEGIN
      GRAPHMODE; TEXT:= FALSE;
    END;
  UNTIL CH IN [ 'Q','q' ];
  TEXTMODE; TEXT:=TRUE;
  TFLAG6:=TRUE;
END;   ( menu  )

BEGIN  ( adv1part  )
(*$R TURTLEGRAPHICS*)
(*$R THESIS0*)
  TESTVALUE:=0;
  TFLAGS:=FALSE;
  MENU;
END;   ( adv1  )
```

```
BEGIN ( case )
  IF TFLAG7 THEN CIRCPART;
  IF TFLAG5 THEN ADV1PART;
END;
SEGMENT PROCEDURE ADVANCE;
VAR TGT: INTEGER;

PROCEDURE MOVEIT;
VAR I,DELTATLEN: INTEGER;
    NEWX,NEWY: PACKED ARRAY [1..10] OF REAL;
    SUCCESSFUL: BOOLEAN;
    SPEED: REAL;

PROCEDURE GETDATA;
VAR J: INTEGER;
BEGIN
  J:=1;
  (*$I-*)
  RESET(ORNGAME, '#5:ORNGNOW.DATA');
  SUCCESSFUL:= (IORESULT=0);
  IF NOT SUCCESSFUL THEN EXIT(GETDATA);
  (*$I+*)
  WHILE (NOT EOF(ORNGAME)) AND (J <= 10) DO BEGIN
    OLDOXY[1,J]:= OXY[1,J];
    OXY[1,J]:=ORNGAME^;
    ORNHIST^:=OXY[1,J]; PUT(ORNHIST);
    GET(ORNGAME);
    OLDOXY[2,J]:= OXY[2,J];
    OXY[2,J]:=ORNGAME^;
    ORNHIST^:=OXY[2,J]; PUT(ORNHIST);
    GET(ORNGAME);
    J:=J + 1;
  END;
  MAXORANGE:= J - 1;
  CLOSE(ORNGAME,LOCK);
  ORNHIST^:= -9999; PUT(ORNHIST);
END;

PROCEDURE CHNGDISK;
BEGIN
  SUCCESSFUL:= TRUE;
  REPEAT
    TEXTMODE; TEXT:= TRUE;
    GOTOXY(0,0); CRT(ERASEOS);
    GOTOXY(0,3);
    WRITELN(' !!!  FOLLOW THESE INSTRUCTIONS  !!! ');
    WRITELN;
    WRITELN;
    IF NOT SUCCESSFUL THEN
    WRITELN('* ERROR HAS OCCURED WHILE TRYING TO READ DATAFILE ON DISK #2 *');
```

```
    WRITELN;
    WRITELN(' 1.  REMOVE GAME DISK FROM DRIVE # 2 ');
    WRITELN;
    WRITELN(' 2.  EXCHANGE DISKS WITH OPPONENT  ');
    WRITELN;
    WRITELN(' 3.  PLACE OPPONENT`S UPDATED GAME DISK IN DRIVE # 2 ');
    WRITELN;
    WRITELN('       (  PRESS RETURN WHEN READY  )  ');
    WRITELN;READLN;
    GETDATA;
  UNTIL SUCCESSFUL;
  ORANGEMOVE:=ORANGEMOVE + 1;
  BLUEMOVE:=BLUEMOVE + 1;
END;

BEGIN  ( moveit )
  TEXTMODE;TEXT:=TRUE;
  DELTATLEN:=2;
  GOTOXY(0,0);CRT(ERASEOS);
  GOTOXY(0,2);
  WRITELN;
  WRITELN('   Enter desired time step ( DELTA T ) for the first move.');
  WRITELN;
  WRITE('        in minutes (0 -- 99):  ');GETINTEGER(DELTAT,DELTATLEN);
  WRITELN;
  IF OLDGAME THEN REWRITE(ATIME, '#4:TIME.DATA');
  OLDGAME:=FALSE;
  ATIME^:=DELTAT; PUT(ATIME);
  REWRITE(BLUGAME, '#5:BLUENOW.DATA');
  FOR I:= 1 TO MAXBLUE DO BEGIN
    SPEED:=GSPD[I]*(DELTAT/60.0);
    NEWX[I]:=BXY[1,I]+(SPEED*2.8*SIN(GCUS[I]));
    OLDBXY[1,I]:= BXY[1,I];
    BXY[1,I]:= NEWX[I];
    BLUGAME^:=NEWX[I];PUT(BLUGAME);
    BLUHIST^:=NEWX[I];PUT(BLUHIST);
    NEWY[I]:=BXY[2,I]+(SPEED*1.92*COS(GCUS[I]));
    OLDBXY[2,I]:= BXY[2,I];
    BXY[2,I]:= NEWY[I];
    BLUGAME^:=NEWY[I];PUT(BLUGAME);
    BLUHIST^:=NEWY[I];PUT(BLUHIST);
  END;
  BLUHIST^:=-9999; PUT(BLUHIST);
  CLOSE(BLUGAME,LOCK);
  CHNGDISK;
  TFLAG6:=TRUE;
  GRAFFLAG:=TRUE;
  TIME:=TIME + DELTAT;
  BLFLAG:=TRUE;
  EXIT(ADVANCE);
```

```
END;

PROCEDURE CHNGDATA(I: INTEGER);
VAR SPD,SPDLEN,CUS,CUSLEN: INTEGER;

BEGIN
  SPDLEN:=2;
  CUSLEN:=3;
  CUS:=ROUND(GCUS[I] * 180 / PI);
  SPD:=ROUND(GSPD[I]);
  WRITELN;WRITELN;
  WRITELN(' ALL NUMERICAL INPUT IS INTEGER!! ');
  WRITELN;
  WRITE(' Course:   (0-360)  ');
  REPEAT
    GETINTEGER(CUS,CUSLEN);
  UNTIL (CUS >= 0) AND (CUS <= 360); WRITELN;
  GCUS[I]:= CUS * PI / 180;
  WRITE(' Speed :   (0-40)   ');
  REPEAT
    GETINTEGER(SPD,SPDLEN);
  UNTIL (SPD >= 0) AND (SPD <= 40);  WRITELN;
  GSPD[I]:= SPD;
END;

PROCEDURE SHOWDATA(I: INTEGER);
VAR YESNO: SETOFCHAR;
    ANS:   CHAR;
BEGIN
  YESNO:=['Y','y','N','n'];
  TEXTMODE; TEXT:= TRUE;
  GOTOXY(0,0); CRT(ERASEOS);
  GOTOXY(0,3);
  WRITE('  Ship  #    : ');WRITE(I-1);WRITELN;
  WRITE(' X Coordinate: ');WRITE(ROUND(BXY[1,I] / 2.8 ));
  WRITELN(' miles ');
  WRITE(' Y Coordinate: ');WRITE(ROUND(BXY[2,I] / 1.92 ));
  WRITELN(' miles ');
  WRITE(' Course      : ');WRITE(ROUND(GCUS[I] * 180 / PI));
  WRITELN(' degrees ');
  WRITE(' Speed       : ');WRITE(ROUND(GSPD[I] ));
  WRITELN(' Knots ');
WRITELN;WRITELN;
WRITE('       Any changes (Y/N) ? ');ANS:= GETCHAR(YESNO + [CHR(13)]);WRITELN;
IF (ANS = 'Y') OR (ANS = 'y') THEN CHNGDATA(I);
END;

PROCEDURE MODALL;
BEGIN
  TEXTMODE; TEXT:= TRUE;
```

```
  FOR TGT:=1 TO MAXBLUE DO BEGIN
    SHOWDATA(TGT);
  END;
END;

PROCEDURE MODONE;
VAR TGTLEN: INTEGER;
BEGIN
  TEXTMODE; TEXT:= TRUE;
  TGTLEN:=1;
  GOTOXY(0,0); CRT(ERASEOS);
  GOTOXY(0,3);
  WRITELN('    Enter number of ship whose parameters you wish to change ');
  WRITELN('                      (0-9) ');
  REPEAT
    GETINTEGER(TGT,TGTLEN);
  UNTIL (TGT >= 0) AND (TGT <= 9);
WRITELN;
SHOWDATA(TGT+1);
END;

PROCEDURE VIEWIT;
VAR
  CH: CHAR;
BEGIN
  TEXTMODE; TEXT:=TRUE;
  REPEAT
    OKSET:=['A'..'B','a'..'b','Q','q'];
    GOTOXY(0,0);CRT(ERASEOS);
    GOTOXY(0,3);
    WRITELN;WRITELN;WRITELN;WRITELN;
    WRITELN('  A)  View / Modify all ships' parameters ');
    WRITELN('  B)  View / Modify one ship's parameters ');
    WRITELN('  Q)  Quit   :  return to previous menu  ');
    WRITELN;WRITELN;WRITELN;
    WRITELN(' ***(  USE "ESC" KEY TO TOGGLE BETWEEN TEXT AND GRAPHICS )*** ');
    CH:=GETCHAR(OKSET+ [CHR(27)]);
    CRT(ERASEOS);
    IF (CH=CHR(27)) AND (TEXT) THEN BEGIN
      GRAFMODE; TEXT:=FALSE;
      CH:='Z';
    END
    ELSE IF (CH=CHR(27)) AND (NOT TEXT) THEN BEGIN
      TEXTMODE; TEXT:=TRUE;
      CH:='Z';
    END;
    CASE CH OF
       'A','a':  MODALL;
       'B','b':  MODONE;
    END;
```

```
  UNTIL CH IN [ 'Q','q' ];
END;   ( viewit )

PROCEDURE MENU;
VAR
  CH: CHAR;
BEGIN
  IF GRAFFLAG THEN BEGIN
    GRAFMODE; TEXT:=FALSE;
    GRAFFLAG:=FALSE;
  END
  ELSE BEGIN
    TEXTMODE; TEXT:=TRUE;
  END;
  REPEAT
    OKSET:=['A'..'B','a'..'b','Q','q'];
    GOTOXY(0,0);CRT(ERASEOS);
    GOTOXY(0,5);
    WRITELN('  A)  View   :  present ship positions, courses, and speeds ');
    WRITELN('  B)  Advance:  move ships to new positions based on time-step');
    WRITELN('  Q)  Quit   :  return to program's outer level ');
    WRITELN;WRITELN;WRITELN;
    WRITELN(' ***(  USE "ESC" KEY TO TOGGLE BETWEEN TEXT AND GRAPHICS )*** ');
    CH:=GETCHAR(OKSET+ [CHR(27),CHR(8),CHR(21),CHR(32)]);
    CRT(ERASEOS);
    IF CH=CHR(8) THEN CH:='R';  ( left arrow ) reduce scale )
    IF CH=CHR(21) THEN CH:='I'; ( right arrow ) increase scale )
    IF (CH=CHR(27)) AND (TEXT) THEN BEGIN
      GRAFMODE; TEXT:=FALSE;
      CH:='Z';
    END
    ELSE IF (CH=CHR(27)) AND (NOT TEXT) THEN BEGIN
      TEXTMODE; TEXT:=TRUE;
      CH:='Z';
    END;
    CASE CH OF
       'A','a': VIEWIT;
       'B','b': MOVEIT;
       'R'    : BEGIN
                  DOWNSCALE;
                  BLFLAG:=TRUE;
                  TFLAG5:=TRUE;
                  TFLAG6:=FALSE;
                  EXIT(ADVANCE);
                END;
       'I'    : BEGIN
                  UPSCALE;
                  BLFLAG:=TRUE;
                  TFLAG5:=TRUE;
                  TFLAG6:=FALSE;
```

```
                EXIT(ADVANCE);
              END;
        ' '   : BEGIN
                RCTRFLAG:=TRUE;
                TFLAG5:=TRUE;
                TFLAG6:=FALSE;
                EXIT(ADVANCE);
              END;
    END;
    IF (CH = 'R') OR (CH = 'I') OR (CH =' ') THEN BEGIN
      GRAFMODE; TEXT:= FALSE;
    END;
  UNTIL CH IN [ 'Q','q' ];
  TEXTMODE; TEXT:=TRUE;
  TFLAG6:=TRUE;
  EXIT(ADVANCE);
END;   ( menu )

BEGIN ( advance )
(*$R TURTLEGRAPHICS*)
(*$R TRANSCEND*)
((*$R THESIS0*))
  TFLAG5:=FALSE;
  TFLAG6:=TRUE;
  MENU;
END;   ( advance )
SEGMENT PROCEDURE CPA;
VAR  TARGET1,TARGET2: CHAR;
   X01,Y01,X11,Y11,DX1,DY1,X02,Y02,X12,Y12,DX2,DY2: REAL;
   BEARING,COURSE1,COURSE2,TGT1,TGT2,I,XX1,XX2,YY1,YY2: INTEGER;
   CPAT,VX1,VX2,VY1,VY2,CUS1,CUS2,T: REAL;
   DIST,DENOM,SPD1,SPD2,DX,DY,DVX,DVY: REAL;

PROCEDURE SPEED(DX,DY: REAL;VAR SPD: REAL);
BEGIN
  DIST:=SQRT(SQR(DX) + SQR(DY));
  SPD:= DIST / DELTAT * 60;
END;

FUNCTION DEGREES(DX,DY: REAL) : INTEGER;
CONST  ERR = 0.01;
VAR VALUE: INTEGER;
BEGIN
  IF (DX < (0+ERR)) AND (DX > (0-ERR)) THEN BEGIN
    IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN DEGREES := 0;
    IF DY > 0 THEN DEGREES := 180;
    IF DY < 0 THEN DEGREES := 360;
    EXIT(DEGREES);
  END;
  IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN BEGIN
```

```
    IF DX > 0 THEN DEGREES := 270;
    IF DX < 0 THEN DEGREES := 090;
    EXIT(DEGREES);
  END;
  VALUE := ROUND(180 / PI * ATAN (DX/DY));
  IF ABS (VALUE) <> VALUE THEN BEGIN
    IF DX < 0 THEN DEGREES:= 180 + VALUE;
    IF DX > 0 THEN DEGREES:= 360 + VALUE;
  END
  ELSE BEGIN
    IF DX > 0 THEN DEGREES:= 180 + VALUE;
    IF DX < 0 THEN DEGREES:= VALUE;
  END;
END;
PROCEDURE TYPE1;
BEGIN
  TGT1:=ORD(TARGET1) - ORD('0');
  TGT1:=TGT1 + 1;
  X01:=OLDBXY[1,TGT1];
  Y01:=OLDBXY[2,TGT1];
  X11:=BXY[1,TGT1];
  Y11:=BXY[2,TGT1];
  DX1:= X01 - X11;
  DY1:= Y01 - Y11;
END;

PROCEDURE TYPE2;
BEGIN
  IF TARGET1 IN ['a'..'j'] THEN TGT1:=ORD(TARGET1) - ORD('a') ELSE
  TGT1:=ORD(TARGET1) - ORD('A');
  TGT1:=TGT1 + 1;
  X01:=OLDOXY[1,TGT1];
  Y01:=OLDOXY[2,TGT1];
  X11:=OXY[1,TGT1];
  Y11:=OXY[2,TGT1];
  DX1:= X01 - X11;
  DY1:= Y01 - Y11;
END;

PROCEDURE TYPE3;
BEGIN
  TGT2:=ORD(TARGET2) - ORD('0');
  TGT2:=TGT2 + 1;
  X02:=OLDBXY[1,TGT2];
  Y02:=OLDBXY[2,TGT2];
  X12:=BXY[1,TGT2];
  Y12:=BXY[2,TGT2];
  DX2:= X02 - X12;
  DY2:= Y02 - Y12;
END;
```

```
PROCEDURE TYPE4;
BEGIN
  IF TARGET2 IN ['a'..'j'] THEN TGT2:=ORD(TARGET2) - ORD('a') ELSE
  TGT2:=ORD(TARGET2) - ORD('A');
  TGT2:=TGT2 + 1;
  X02:=OLDOXY[1,TGT2];
  Y02:=OLDOXY[2,TGT2];
  X12:=OXY[1,TGT2];
  Y12:=OXY[2,TGT2];
  DX2:= X02 - X12;
  DY2:= Y02 - Y12;
END;

PROCEDURE DRAWLINES;
VAR X1,Y1,X2,Y2,DX,DY,SPD3: REAL;
BEGIN
  GRAPHMODE; TEXT:= FALSE;
  PENCOLOR(NONE);
  XX1:=ROUND((X11-XCTR)/SCALE)+140;
  YY1:=ROUND((Y11-YCTR)/SCALE)+96;
  MOVETO(XX1,YY1);
  PENCOLOR(WHITE);
  XX1:=ROUND(XX1+(T*VX1*2.8/SCALE));
  YY1:=ROUND(YY1+(T*VY1*1.92/SCALE));
  X1:=((XX1-140)*SCALE)+XCTR;
  Y1:=((YY1-96)*SCALE)+YCTR;
  MOVETO(XX1,YY1);
  PENCOLOR(NONE);
  XX2:=ROUND((X12-XCTR)/SCALE)+140;
  YY2:=ROUND((Y12-YCTR)/SCALE)+96;
  MOVETO(XX2,YY2);
  PENCOLOR(WHITE);
  XX2:=ROUND(XX2+(T*VX2*2.8/SCALE));
  YY2:=ROUND(YY2+(T*VY2*1.92/SCALE));
  X2:=((XX2-140)*SCALE)+XCTR;
  Y2:=((YY2-96)*SCALE)+YCTR;
  MOVETO(XX2,YY2);
  PENCOLOR(NONE);
  DX:=(X1-X2) / 2.8;
  DY:=(Y1-Y2) / 1.92;
  BEARING:= DEGREES(DX,DY);
  SPEED(DX,DY,SPD3);
  FOR I:= 1 TO 3000 DO BEGIN
    I:=I+1;
  END;
END;

PROCEDURE MESSAGE1;
BEGIN
```

```
  TEXTMODE; TEXT:= TRUE;
  GOTOXY(0,3);CRT(ERASEOS);
  GOTOXY(0,6);
  WRITELN('   Not enough data to compute CPA ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('    < PRESS RETURN TO CONTINUE >   ');
  READLN;
  TFLAG3:=FALSE;
  TFLAG4:= TRUE;
  EXIT(CPA);
END;

PROCEDURE MESSAGE2;
BEGIN
  TEXTMODE; TEXT:= TRUE;
  GOTOXY(0,0);CRT(ERASEOS);
  GOTOXY(0,6);
  WRITELN('    The time to CPA from ',TARGET1,' to ',TARGET2,' is  : ');
  WRITELN;
  WRITELN('            ',ROUND(T*60),' MINUTES ');
  WRITELN;
  WRITELN('    The range and bearing from ',TARGET1,' to ',TARGET2);
  WRITELN('                  at that time will be :  ');
  WRITELN;
  WRITELN('           ',ROUND(DIST),' miles -- bearing ',BEARING,' degrees ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('       < PRESS RETURN TO CONTINUE >   ');
  READLN;
  TFLAG3:=FALSE;
  TFLAG4:=TRUE:
END;

PROCEDURE MESSAGE3;
BEGIN
  TEXTMODE: TEXT:= TRUE;
  GOTOXY(0,3);CRT(ERASEOS);
  GOTOXY(0,6);
  WRITELN('    Target # ',TARGET1,' and target # ',TARGET2 );
  WRITELN;
  WRITELN('    have parallel courses with the same speed ! ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('       < PRESS RETURN TO CONTINUE >   ');
  READLN;
  TFLAG3:=FALSE;
  TFLAG4:=TRUE;
  EXIT(CPA):
END;


BEGIN ( CPA )
```

```
(*$R TURTLEGRAPHICS*)
(*$R TRANSCEND*)
(*$R THESIS0*)
  IF ORANGEMOVE (= 1 THEN MESSAGE1;
  GRAFMODE; TEXT:=FALSE;
  OKSET:=['0'..'9','A'..'J','a'..'j'];
  BOXSTRING(3,2,'   CPA:  LETTER / NUMBER OF 1ST SHIP   ');
  TARGET1:=GETCHAR(OKSET);
  IF TARGET1 IN['0'..'9']THEN TYPE1 ELSE TYPE2;
  DX1:=DX1 / 2.8;
  DY1:=DY1 / 1.92;
  COURSE1:=DEGREES(DX1,DY1);
  CUS1:=COURSE1 * PI / 180;
  SPEED(DX1,DY1,SPD1);
  VX1:=SPD1 * SIN(CUS1);
  VY1:=SPD1 * COS(CUS1);
  BOXSTRING(3,2,'   CPA:   LETTER / NUMBER OF 2ND SHIP  ');
  TARGET2:=GETCHAR(OKSET);
  CLEARBOX;
  IF TARGET2 IN['0'..'9']THEN TYPE3 ELSE TYPE4;
  DX2:=DX2 / 2.8;
  DY2:=DY2 / 1.92;
  COURSE2:=DEGREES(DX2,DY2);
  CUS2:=COURSE2 * PI / 180;
  SPEED(DX2,DY2,SPD2);
  VX2:=SPD2 * SIN(CUS2);
  VY2:=SPD2 * COS(CUS2);
  DX:=(X11 - X12) / 2.8;
  DY:=(Y11 - Y12) / 1.92;
  DVX:=VX1 - VX2;
  DVY:=VY1 - VY2;
  DENOM:= SQR(DVX) + SQR(DVY);
  IF DENOM = 0 THEN MESSAGE3;
  T:= -(DX*DVX + DY*DVY) / (SQR(DVX) + SQR(DVY));
  IF T < 0 THEN T:= 0;
  CPAT:= TIME + T;
  DRAWLINES;
  MESSAGE2;
END;

SEGMENT PROCEDURE MOVEMENT;
VAR  CH: CHAR;
     X,Y,TYME,TARGET,MCUS,DT: INTEGER;
     DIRECTION,COURSE,SPEED,MSPD: REAL;
     NOTYET,TRACE: BOOLEAN;

PROCEDURE DISPLAY;
VAR X,Y,J: INTEGER;
BEGIN
  INITTURTLE;
```

```
  FOR J:=1 TO MAXBLUE DO BEGIN
    X:=ROUND((BXY[1,J]-XCTR)/SCALE)+140;
    Y:=ROUND((BXY[2,J]-YCTR)/SCALE)+96;
    MOVETO(X,Y);
    DRAWBLOCK(SHIPFIG,2,0,0,7,8,X,Y,10);
    NUMBER(X,Y,J);
  END;
  FOR J:= 1 TO MAXORANGE DO BEGIN
    X:=ROUND((OXY[1,J]-XCTR)/SCALE)+140;
    Y:=ROUND((OXY[2,J]-YCTR)/SCALE)+96;
    MOVETO(X,Y);
    DRAWBLOCK(SHIPFIG,2,0,0,7,3,X,Y,5);
    LETTER(X,Y,J);
  END;
  GRAFMODE; TEXT:=FALSE;
END;
PROCEDURE MESSAGE1;
BEGIN
  CLEARBOX;
  GOTOXY(0,0); CRT(ERASEOS);
  GOTOXY(0,6);
  WRITELN('  Not enough data to compute course / speed ');
  WRITELN;WRITELN;WRITELN;
  WRITELN('     < PRESS RETURN TO CONTINUE >  ');
  TEXTMODE; TEXT:=TRUE;
  NOTYET:=TRUE;
  READLN;
END;

PROCEDURE GETTGT;
VAR TGT: CHAR;
BEGIN
  OKSET:=['0'..'9','A'..'J','a'..'j'];
  BOXSTRING(3,2,' MOVEMENT:  MOVE WHICH SHIP (0-9/A-J)? ');
  TGT:=GETCHAR(OKSET);
  IF (TGT IN ['A'..'J']) OR (TGT IN ['a'..'j']) THEN BEGIN
    IF ORANGEMOVE <= 1 THEN MESSAGE1;
    IF ORANGEMOVE <= 1 THEN EXIT(GETTGT);
    IF TGT IN ['A'..'J'] THEN MTARGET:=ORD(TGT)-ORD('A')+1 ELSE
    MTARGET:=ORD(TGT)-ORD('a')+1;
    MMFLAG:=TRUE;
    TFLAG2:=FALSE;
    EXIT(MOVEMENT);
  END;
  TARGET:=ORD(TGT) - ORD('0') + 1;
END;
PROCEDURE GETCOURSE;
VAR CUSLEN,CUS: INTEGER;
BEGIN
  CUSLEN:=3;
```

```
  REPEAT
    BOXSTRING(3,2,' MOVEMENT:ENTER DESIRED COURSE (0-360) ');
    GETINTEGER(CUS,CUSLEN);
  UNTIL (CUS >= 0) AND (CUS <= 360);
  MCUS:=CUS;
  COURSE:=CUS * PI / 180;
END;
PROCEDURE GETSPEED;
VAR SPDLEN,SPD: INTEGER;
BEGIN
  SPDLEN:=2;
  REPEAT
    BOXSTRING(3,2,' MOVEMENT:  ENTER DESIRED SPEED (0-40) ');
    GETINTEGER(SPD,SPDLEN);
  UNTIL (SPD >= 0) AND (SPD <= 40);
  MSPD:=SPD;
  SPEED:=SPD;
END;
PROCEDURE GETTIME;
VAR TIMELEN: INTEGER:
BEGIN
  TIMELEN:=2;
  REPEAT
    BOXSTRING(3,2,' MOVEMENT: TIME STEP (0-99) IN MINUTES ');
    GETINTEGER(TYME,TIMELEN);
  UNTIL (TYME >= 0) AND (TYME <= 99);
END;
PROCEDURE TRACEON;
VAR TR: CHAR;
BEGIN
  OKSET:=['Y','y','N','n'];
  BOXSTRING(3,2,' MOVEMENT: DO YOU WISH TRACE ON (Y/N)? ');
  TR:=GETCHAR(OKSET + [CHR(13)]);
  IF (TR = 'Y') OR (TR = 'y') THEN TRACE:=TRUE ELSE TRACE:=FALSE;
END;
PROCEDURE GETX;
VAR XLEN: INTEGER;
    X1: REAL;
BEGIN
  XLEN:=3;
  REPEAT
    BOXSTRING(3,2,'MOVEMENT: DESIRED X-COORDINATE (0-100)?');
    GETINTEGER(X,XLEN);
  UNTIL (X >= 0) AND (X <= 100);
  X1:= X * 2.8;
  X1:=((X1-XCTR)/SCALE)+140;
  X:=ROUND(X1);
END;
PROCEDURE GETY;
VAR YLEN: INTEGER;
```

```
    Y1: REAL;
BEGIN
  YLEN:=3;
  REPEAT
    BOXSTRING(3,2,'MOVEMENT: DESIRED Y-COORDINATE (0-100)?');
    GETINTEGER(Y,YLEN);
  UNTIL (Y >= 0) AND (Y <= 100);
  Y1:= Y * 1.92;
  Y1:=((Y1-YCTR)/SCALE)+96;
  Y:=ROUND(Y1);
END;
PROCEDURE RELDIR;
VAR DIRLEN,DIR: INTEGER;
BEGIN
  DIRLEN:=3;
  REPEAT
    BOXSTRING(3,2,' MOVEMENT:  DESIRED DIRECTION (0-360)? ');
    GETINTEGER(DIR,DIRLEN);
  UNTIL (DIR >= 0) AND (DIR <= 360);
  MCUS:=DIR;
  DIRECTION:= DIR * PI /180;
END;
PROCEDURE RELDIST;
VAR DISTLEN,DIST: INTEGER;
    XD,YD: REAL;
BEGIN
  DISTLEN:=3;
  REPEAT
    BOXSTRING(3,2,'MOVEMENT:DESIRED DISTANCE (0-300) MILES');
    GETINTEGER(DIST,DISTLEN);
  UNTIL (DIST >= 0) AND (DIST <= 300);
  IF DELTAT = 0 THEN DT:=60 ELSE DT:=DELTAT;
  MSPD:=GSPD[TARGET];
  XD:= DIST * SIN(DIRECTION);
  XD:= XD * 2.30 / SCALE;
  X:= ROUND(XD);
  YD:= DIST * COS(DIRECTION);
  YD:= YD * 1.92 / SCALE;
  Y:= ROUND(YD);
END;
PROCEDURE DRAWFIG;
BEGIN
  PENCOLOR(NONE);
  DRAWBLOCK(SHIPFIG,2,0,0,7,3,X,Y,10);
  NUMBER(X,Y,TARGET);
END;
PROCEDURE MAKECHNG(C : INTEGER;S : REAL);
VAR I: INTEGER;
    ANS: CHAR;
BEGIN
```

```
  OKSET:=['Y','y','N','n'];
  FOR I:=1 TO 3000 DO;
  TEXTMODE; TEXT:=TRUE;
  GOTOXY(0,0); CRT(ERASEOS);
  GOTOXY(0,3);
  WRITELN(' YOU HAVE PROPOSED A COURSE & SPEED CHANGE FOR TARGET# ',TARGET-1);
  WRITELN;
  WRITELN('        ',C,' DEGREES   ',ROUND(S),' KNOTS ');
  WRITELN;
  WRITELN('    DO YOU WISH TO MAKE THIS CHANGE TO THE DATABASE ? (Y/N) ');
  ANS:=GETCHAR(OKSET + [CHR(13)]);
  IF ANS IN ['Y','y'] THEN BEGIN
    GCUS[TARGET]:=CXPI/180;
    GSPD[TARGET]:=S;
  END;
END;

PROCEDURE MOVE1;
VAR XD,YD: REAL;
    B1,B2: INTEGER;
BEGIN
  GRAFMODE; TEXT:= FALSE;
  GETTGT;
  IF NOTYET THEN BEGIN
    NOTYET:=FALSE;
    EXIT(MOVE1);
  END;
  GETCOURSE;
  GETSPEED;
  GETTIME;
  TRACEON;
  CLEARBOX;
  PENCOLOR(NONE);
  B1:=ROUND((BXY[1,TARGET]-XCTR)/SCALE)+140;
  B2:=ROUND((BXY[2,TARGET]-YCTR)/SCALE)+96;
  MOVETO(B1,B2);
  {  do math to get new xy  }
  IF TRACE THEN PENCOLOR(WHITE);
  XD:= SPEED / 60 * TYME * SIN(COURSE) / SCALE * 2.8;
  X:= ROUND(XD);
  X:=X+B1;
  YD:= SPEED / 60 * TYME * COS(COURSE) / SCALE * 1.92;
  Y:= ROUND(YD);
  Y:=Y+B2;
  MOVETO(X,Y);
  DRAWFIG;
  MAKECHNG(MCUS,MSPD);
END;

PROCEDURE MOVE2;
```

```
VAR B1,B2: INTEGER;
    DX,DY: REAL;
BEGIN
  GRAFMODE; TEXT:=FALSE;
  GETTGT;
  IF NOTYET THEN BEGIN
    NOTYET:=FALSE;
    EXIT(MOVE2);
  END;
  GETX;
  GETY;
  TRACEON;
  CLEARBOX;
  PENCOLOR(NONE);
  B1:=ROUND((BXY[1,TARGET]-XCTR)/SCALE)+140;
  B2:=ROUND((BXY[2,TARGET]-YCTR)/SCALE)+96;
  MOVETO(B1,B2);
  IF TRACE THEN PENCOLOR(WHITE);
  MOVETO(X,Y);
  DRAWFIG;
END;

PROCEDURE MOVE3;
VAR B1,B2: INTEGER;
BEGIN
  GRAFMODE; TEXT:=FALSE;
  GETTGT;
  IF NOTYET THEN BEGIN
    NOTYET:=FALSE;
    EXIT(MOVE3);
  END;
  RELDIR;
  RELDIST;
  TRACEON;
  CLEARBOX;
  PENCOLOR(NONE);
  B1:=ROUND((BXY[1,TARGET]-XCTR)/SCALE)+140;
  B2:=ROUND((BXY[2,TARGET]-YCTR)/SCALE)+96;
  MOVETO(B1,B2);
  IF TRACE THEN PENCOLOR(WHITE);
  X:=X+B1;
  Y:=Y+B2;
  MOVETO(X,Y);
  DRAWFIG;
  MAKECHNG(MCUS,MSPD);
END;

PROCEDURE RESETFLAGS;
BEGIN
  NOTYET:=FALSE;
```

```
  MMFLAG:=FALSE;
  TFLAG1:=FALSE;
  TFLAG2:= TRUE;
  IF GRAFFLAG THEN
  BEGIN
    GRAFMODE;
    GRAFFLAG:=FALSE;
    TEXT:=FALSE;
  END
  ELSE BEGIN
    TEXTMODE;
    TEXT:=TRUE;
  END;
END;

BEGIN   { movement }
(*$R TURTLEGRAPHICS*)
(*$R THESIS0*)
(*$R TRANSCEND*)
  RESETFLAGS;
  REPEAT
    OKSET:=['A'..'D','a'..'d','Q','q'];
    GOTOXY(0,0); CRT(ERASEOS);
    PROMPTAT(0,') MOVEMENT: Visually check possible positions of your ships');
    GOTOXY(0,5);
    WRITELN('   A)  Movement using Course, Speed, Time ');
    WRITELN('   B)  Move your unit to a new (X,Y) position ');
    WRITELN('   C)  Move your unit relative to its present position ');
    WRITELN('   D)  Clear previous movements from the screen ');
    WRITELN('   Q)  Quit - Exit from this procedure ');
    WRITELN;
    WRITELN(' [<-]    Downscale display ');
    WRITELN(' [->]    Upscale display ');
    WRITELN('[space]  Recenters display ');
    WRITELN(' [ESC]   Toggles display between text and graphics ');
    CH:=GETCHAR(OKSET+ [CHR(27),CHR(8),CHR(21),CHR(32)]);
    CRT(ERASEOS);
    IF CH=CHR(8) THEN CH:='R';  { left arrow } reduce scale }
    IF CH=CHR(21) THEN CH:='I';  { right arrow } increase scale }
    IF (CH=CHR(27)) AND (TEXT) THEN BEGIN
      GRAFMODE;
      TEXT:=FALSE;
      CH:='Z';
    END
    ELSE IF (CH=CHR(27)) AND (NOT TEXT) THEN BEGIN
      TEXTMODE;
      TEXT:=TRUE;
      CH:='Z';
    END;
    CASE CH OF
```

```
      'A','a':  MOVE1;
      'B','b':  MOVE2;
      'C','c':  MOVE3;
      'D','d':  DISPLAY;
      'R'    :  BEGIN
                  DOWNSCALE;
                  BLFLAG:=TRUE;
                  TFLAG1:=TRUE;
                  TFLAG2:=FALSE;
                  EXIT(MOVEMENT);
                END;
      'I'    :  BEGIN
                  UPSCALE;
                  BLFLAG:=TRUE;
                  TFLAG1:=TRUE;
                  TFLAG2:=FALSE;
                  EXIT(MOVEMENT);
                END;
      ' '    :  BEGIN
                  RCTRFLAG:=TRUE;
                  BLFLAG:=TRUE;
                  TFLAG1:=TRUE;
                  TFLAG2:=FALSE;
                  EXIT(MOVEMENT);
                END;
    END;
  UNTIL CH IN [ 'Q','q' ];
END;

SEGMENT PROCEDURE MOVE2;
VAR X1,Y1,X2,Y2,DISTANCE,SPEED,CUS,DX,DY: REAL;
    COURSE,OD1,OD2,TYME: INTEGER;
    TRACE: BOOLEAN;
    XX,YY: REAL;

PROCEDURE DISTSPD(DX,DY: REAL;VAR DIST,SPD: REAL);
BEGIN
  DIST:= SQRT(SQR(DX) + SQR(DY));
  SPD:= DIST / DELTAT;
END;

FUNCTION DEGREES(DX,DY: REAL) : INTEGER;
CONST  ERR = 0.01;
VAR VALUE: INTEGER;
BEGIN
  IF (DX < (0+ERR)) AND (DX > (0-ERR)) THEN BEGIN
    IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN DEGREES := 0;
    IF DY > 0 THEN DEGREES := 180;
    IF DY < 0 THEN DEGREES := 360;
    EXIT(DEGREES);
```

```
  END;
  IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN BEGIN
    IF DX > 0 THEN DEGREES := 270;
    IF DX < 0 THEN DEGREES := 090;
    EXIT(DEGREES);
  END;
  VALUE := ROUND(180 / PI * ATAN (DX/DY));
  IF ABS (VALUE) <> VALUE THEN BEGIN
    IF DX < 0 THEN DEGREES:= 180 + VALUE;
    IF DX > 0 THEN DEGREES:= 360 + VALUE;
  END
  ELSE BEGIN
    IF DX > 0 THEN DEGREES:= 180 + VALUE;
    IF DX < 0 THEN DEGREES:= VALUE;
  END;
END;

PROCEDURE GETTIME;
VAR TIMELEN: INTEGER;
BEGIN
  TIMELEN:=2;
  REPEAT
    BOXSTRING(3,2,'MOVEMENT2: TIME STEP (0-99) IN MINUTES');
    GETINTEGER(TYME,TIMELEN);
  UNTIL (TYME >= 0) AND (TYME <= 99);
END;

PROCEDURE TRACEON;
VAR TR: CHAR;
BEGIN
  OKSET:=['Y','y','N','n'];
  BOXSTRING(3,2,'MOVEMENT2: DO YOU WANT TRACE ON (Y/N)?');
  TR:=GETCHAR(OKSET + [CHR(13)]);
  IF (TR = 'Y') OR (TR ='y') THEN TRACE:=TRUE ELSE TRACE:=FALSE;
END;

PROCEDURE NEWPOSIT;
VAR NEWX,NEWY: REAL;
BEGIN
  XX:=(OXY[1,MTARGET]-XCTR)/SCALE+140;
  YY:=(OXY[2,MTARGET]-YCTR)/SCALE+96;
  NEWX:= XX + (SPEED * TYME * SIN(CUS) * 2.3 / SCALE);
  NEWY:= YY + (SPEED * TYME * COS(CUS) * 1.92 / SCALE);
  001:=ROUND(NEWX);
  002:=ROUND(NEWY);
END;

PROCEDURE MOVEIT;
BEGIN
  GRAFMODE; TEXT:=FALSE;
```

```
  CLEARBOX;
  PENCOLOR(NONE);
  MOVETO(ROUND(XX),ROUND(YY));
  IF TRACE THEN PENCOLOR(WHITE);
  MOVETO(001,002);
  PENCOLOR(NONE);
  DRAWBLOCK(SHIPFIG,2,0,0,7,8,001,002,5);
  LETTER(001,002,MTARGET);
END;

BEGIN ( move2 )
(*$R TURTLEGRAPHICS*)
(*$R THESIS0*)
(*$R TRANSCEND*)
  GRAFMODE; TEXT:=FALSE;
  MMFLAG:=FALSE;
  TFLAG1:=TRUE;
  X1:=OLDXY[1,MTARGET];
  Y1:=OLDXY[2,MTARGET];
  X2:=XY[1,MTARGET];
  Y2:=XY[2,MTARGET];
  DX:=(X1 - X2) / 2.8;
  DY:=(Y1 - Y2) / 1.92;
  COURSE:=DEGREES(DX,DY);
  CUS:= COURSE * PI / 180;
  DISTSPD(DX,DY,DISTANCE,SPEED);
  GETTIME;
  TRACEON;
  NEWPOSIT;
  MOVEIT;
END; ( move2 )
SEGMENT PROCEDURE RECENTER;

PROCEDURE RCTRPART;
CONST  XC=140;
       YC=96;
VAR J,BX,BY,CX,CY: INTEGER;
    RCENTR: CHAR;

PROCEDURE CASEA;
BEGIN
  J:=ORD(RCENTR)-ORD('0');
  J:=J+1;
  BX:=ROUND((BXY[1,J1-XC)/SCALE)+XC;
  BY:=ROUND((BXY[2,J1-YC)/SCALE)+YC;
  XCTR:=ROUND(BXY[1,J1);
  YCTR:=ROUND(BXY[2,J1);
END;

PROCEDURE CASEB;
```

```
BEGIN
IF RCENTR IN ['A'..'J'] THEN J:=ORD(RCENTR)-ORD('A')
  ELSE J:=ORD(RCENTR)-ORD('a');
  J:=J+1;
  OX:=ROUND((OXY[1,J]-XC)/SCALE)+XC;
  OY:=ROUND((OXY[2,J]-YC)/SCALE)+YC;
  XCTR:=ROUND(OXY[1,J]);
  YCTR:=ROUND(OXY[2,J]);
END;

BEGIN ( recenter )
(*$R THESIS0*)
(*$R TURTLEGRAPHICS*)
  RCTRFLAG:=FALSE;
  BLFLAG:=TRUE;
  GRAFMODE; TEXT:=FALSE;
  IF (MAXORANGE > 0) AND (MAXBLUE > 0) THEN BEGIN
    OKSET:=['0'..'9','A'..'J','a'..'j'];
    BOXSTRING(3,2,'  RECENTER ON WHICH SHIP (0-9/A-J)?   ');
    RCENTR:=GETCHAR(OKSET);
    IF RCENTR IN ['0'..'9'] THEN CASEA ELSE CASEB;
  END;
  IF (MAXBLUE = 0) THEN
  BEGIN
    OKSET:=['0'..'9'];
    BOXSTRING(3,2,'    RECENTER ON WHICH SHIP (0-9)?     ');
    RCENTR:=GETCHAR(OKSET);
    CASEB;
  END;
  IF (MAXORANGE = 0) THEN
  BEGIN
    OKSET:=['0'..'9'];
    BOXSTRING(3,2,'    RECENTER ON WHICH SHIP (0-9)?     ');
    RCENTR:=GETCHAR(OKSET);
    CASEA;
  END;
  INITTURTLE;
  IF MAXBLUE > 0 THEN BEGIN
    FOR J:=1 TO MAXBLUE DO BEGIN
      BX:=ROUND((BXY[1,J]-XCTR)/SCALE)+XC;
      BY:=ROUND((BXY[2,J]-YCTR)/SCALE)+YC;
      DRAWBLOCK(SHIPFIG,2,0,0.7,3,BX,BY,10);
      NUMBER(BX,BY,J)
    END;
  END;
  IF MAXORANGE > 0 THEN BEGIN
    FOR J:=1 TO MAXORANGE DO BEGIN
      OX:=ROUND((OXY[1,J]-XCTR)/SCALE)+XC;
      OY:=ROUND((OXY[2,J]-YCTR)/SCALE)+YC;
      DRAWBLOCK(SHIPFIG,2,0,0.7,3,OX,OY,5);
```

```
      LETTER(OX,OY,J)
    END;
  END;
END;

PROCEDURE RERUNPART;
VAR J: INTEGER;
    TESTVALUE: REAL;

PROCEDURE STARTORANGE;
BEGIN
  J:=1;
  TESTVALUE:=0;
  RESET(ORNHIST, '#4:ORNGOLD.DATA');
  WHILE (NOT EOF(ORNHIST)) AND (TESTVALUE > -999) AND (J <= 18) DO BEGIN
    OXY[1,J]:=ORNHIST^;
    GET(ORNHIST);
    OXY[2,J]:=ORNHIST^;
    J:=J+1;
    GET(ORNHIST);
    TESTVALUE:=ORNHIST^;
    MAXORANGE:=J - 1;
  END;
END;

PROCEDURE STARTBLUE;
BEGIN
  J:=1;
  TESTVALUE:=0;
  RESET(BLUHIST, '#4:BLUEOLD.DATA');
  WHILE (NOT EOF(BLUHIST)) AND (TESTVALUE > -999) AND (J <= 18) DO BEGIN
    BXY[1,J]:=BLUHIST^;
    GET(BLUHIST);
    BXY[2,J]:=BLUHIST^;
    J:=J+1;
    GET(BLUHIST);
    TESTVALUE:=BLUHIST^;
    MAXBLUE:=J - 1;
  END;
END;

PROCEDURE RESETFLAGS;
BEGIN
  CONTINUE:=FALSE;
  DBCALLED:=FALSE;
  CASE1:=FALSE;
  TFLAG1:=FALSE;
  TFLAG2:=FALSE;
  TFLAG3:=FALSE;
  TFLAG4:=FALSE;
```

```
  TFLAG5:=FALSE;
  TFLAG6:=FALSE;
  TEXT:=TRUE;
END;

BEGIN  ( RERUN )
  TESTVALUE:=0;
  RRFLAG:=TRUE;
  RESETFLAGS;
  CONTINUE:=TRUE;
  BLFLAG:=TRUE;
  MAXORANGE:=0;
  ORANGEMOVE:=0;
  BLUEMOVE:=0;
  SCALE:=1;
  INITTURTLE; TEXTMODE;
  CLOSE(BLUHIST,LOCK);
  CLOSE(ORNHIST,LOCK);
  STARTORANGE;
  ORANGEMOVE:=ORANGEMOVE + 1;
  STARTBLUE;
  BLUEMOVE:=BLUEMOVE + 1;
  RESET(ATIME, '#4:TIME.DATA');
  TIME:=ATIME^;
END;  ( rerun )


BEGIN ( CASE )
  IF RCTRFLAG THEN RCTRPART;
  IF CASE1 THEN RERUNPART;
END;
SEGMENT PROCEDURE BOTTOMLINE;
VAR SCDIST: REAL;
    XT,XC,YC: INTEGER;
    STIME,SXCTR,SYCTR: STRING;

PROCEDURE HEADER;
BEGIN
(*$R TURTLEGRAPHICS*)
  GRAFMODE; TEXT:=FALSE;
  BLFLAG:=FALSE;
  GRAFFLAG:=TRUE;
  PENCOLOR(NONE);
  MOVETO(0,184);
  WSTRING('                                        ');
  MOVETO(0,184);
  PENCOLOR(WHITE);
  MOVETO(0,190);
  MOVETO(0,187);
  SCDIST:= 10/SCALE*2.72;
```

```
  XT:=ROUND(SCDIST);
  MOVETO(XT,187);
  MOVETO(XT,190);
  MOVETO(XT,184);
  PENCOLOR(NONE);
  MOVETO(XT+7,182);
  WSTRING(': 10 NM  ');
  XC:=ROUND(XCTR/2.8);
  YC:=ROUND(YCTR/1.92);
  STR(XC,SXCTR);
  STR(YC,SYCTR);
  WSTRING(SXCTR);WSTRING(',');WSTRING(SYCTR);
  WSTRING(': CTR ');
  WSTRING(' TIME: ');
  STR(TIME,STIME);
  WSTRING(STIME);
END;

PROCEDURE INTERCEPT;
VAR  TGT,TGT2,CUS1,DT,DTLEN,ICUS: INTEGER;
     T1,T2: CHAR;
     COURSE1,SPEED1,VX1,VY1,X,Y,IX,IY,DIX,DIY,ISPD,DX1,DY1,X2,Y2: REAL;
PROCEDURE TYPE1(T: CHAR; VAR DX,DY: REAL);
VAR X1,Y1: REAL;
BEGIN
  TGT:=ORD(T)-ORD('0')+1;
  X1:=OLDBXY[1,TGT]/2.8;
  Y1:=OLDBXY[2,TGT]/1.92;
  X2:=BXY[1,TGT]/2.8;
  Y2:=BXY[2,TGT]/1.92;
  DX:=(X1-X2);
  DY:=(Y1-Y2);
END;
PROCEDURE TYPE2(T: CHAR; VAR DX,DY: REAL);
VAR X1,Y1: REAL;
BEGIN
  IF T IN ['a'..'j'] THEN TGT:=ORD(T)-ORD('a')+1 ELSE TGT:=ORD(T)-ORD('A')+1;
  X1:=OLDOXY[1,TGT]/2.8;
  Y1:=OLDOXY[2,TGT]/1.92;
  X2:=OXY[1,TGT]/2.8;
  Y2:=OXY[2,TGT]/1.92;
  DX:=(X1-X2);
  DY:=(Y1-Y2);
END;

FUNCTION DEGREES(DX,DY: REAL) : INTEGER;
CONST  ERR = 0.01;
VAR VALUE: INTEGER;
BEGIN
  IF (DX < (0+ERR)) AND (DX > (0-ERR)) THEN BEGIN
```

```
    IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN DEGREES := 0;
    IF DY > 0 THEN DEGREES := 180;
    IF DY < 0 THEN DEGREES := 360;
    EXIT(DEGREES);
  END;
  IF (DY < (0+ERR)) AND (DY > (0-ERR)) THEN BEGIN
    IF DX > 0 THEN DEGREES := 270;
    IF DX < 0 THEN DEGREES := 090;
    EXIT(DEGREES);
  END;
  VALUE := ROUND(180 / PI * ATAN (DX/DY));
  IF ABS (VALUE) <> VALUE THEN BEGIN
    IF DX < 0 THEN DEGREES:= 180 + VALUE;
    IF DX > 0 THEN DEGREES:= 360 + VALUE;
  END
  ELSE BEGIN
    IF DX > 0 THEN DEGREES:= 180 + VALUE;
    IF DX < 0 THEN DEGREES:= VALUE;
  END;
END;

PROCEDURE MESSAGE;
BEGIN
  GOTOXY(0,0);CRT(ERASEOS);
  GOTOXY(0,3);
  TEXTMODE; TEXT:=TRUE;
  WRITELN('  NOT ENOUGH DATA TO COMPUTE INTERCEPTS AT THIS TIME ');
  WRITELN;
  WRITELN('     (   PRESS RETURN TO CONTINUE  )'); READLN;
  EXIT(BOTTOMLINE);
END;

PROCEDURE MESSAGE2;
BEGIN
  GOTOXY(0,0);CRT(ERASEOS);
  GOTOXY(0,3);
  TEXTMODE; TEXT:=TRUE;
  WRITELN(' INTERCEPT:  THE COURSE AND SPEED REQUIRED FOR UNIT# ',T2);
  WRITELN('             TO INTERCEPT UNIT# ',T1,' IS: ');
  WRITELN;
  WRITELN('            ', ICUS,' DEGREES AT ',ROUND(ISPD),' KNOTS ');
  WRITELN;
  WRITELN('          DO YOU WANT TO MAKE THIS CHANGE TO THE DATABASE? (Y/N)');
  OKSET:=['Y','y','N','n'];
  T1:=GETCHAR(OKSET + [CHR(13)]);
  IF T1 IN ['Y','y'] THEN BEGIN
    GCUSITGT2]:=ICUS*PI/180;
    GSPD[TGT2]:=ISPD;
    WRITELN('             CHANGE HAS BEEN MADE ! ');
    WRITELN;
```

```
    WRITELN('      (  PRESS RETURN TO CONTINUE  )   ');READLN;
  END;
END;

BEGIN
(*$R THESIS0*)
(*$R TRANSCEND*)
(*$R TURTLEGRAPHICS*)
  ICPTFLAG:=FALSE;
  BLFLAG:=FALSE;
  IF (BLUEMOVE <= 1) OR (TIME = 0) THEN MESSAGE;
  GRAFMODE; TEXT:=FALSE;
  OKSET:=['0'..'9','A'..'J','a'..'j'];
  DTLEN:=3;
  BOXSTRING(3,2,'ICPT:LETTER/NUMBER OF UNIT TO INTERCEPT');
  T1:=GETCHAR(OKSET);
  IF T1 IN ['0'..'9'] THEN TYPE1(T1,DX1,DY1) ELSE TYPE2(T1,DX1,DY1);
  CUS1:=DEGREES(DX1,DY1);
  COURSE1:=CUS1 * PI / 180;
  SPEED1:=SQRT(SQR(DX1)+SQR(DY1))/DELTAT;
  VX1:=SPEED1*SIN(COURSE1);
  VY1:=SPEED1*COS(COURSE1);
  OKSET:=['0'..'9'];
  BOXSTRING(3,2,' ICPT:NUMBER OF MANEUVERING UNIT (0-9) ');
  T2:=GETCHAR(OKSET);
  TGT2:=ORD(T2)-ORD('0')+1;
  X:=BXY[1,TGT2]/2.8;
  Y:=BXY[2,TGT2]/1.92;
  REPEAT
    BOXSTRING(3,2,'ICPT: DESIRED TIME TO INTERCEPT (0-999)');
    GETINTEGER(DT,DTLEN);
  UNTIL(DT >= 0) AND (DT <= 999);
  CLEARBOX;
  IX:=(VX1*DT)+X2;
  IY:=(VY1*DT)+Y2;
  DIX:=X-IX;
  DIY:=Y-IY;
  ICUS:=DEGREES(DIX,DIY);
  ISPD:=SQRT(SQR(DIX)+SQR(DIY))/DT*60;
  MESSAGE2;
END;

BEGIN ( bline )
  IF ICPTFLAG THEN INTERCEPT;
  IF BLFLAG THEN HEADER;
END;
 SEGMENT PROCEDURE TMAFINIT;
 VAR GFLAG,GFLAG1: CHAR;
     LEVEL3: SETOFCHAR;
     SAVEFLAG: BOOLEAN;
```

```
    BNOW,BOLD,ONOW,OOLD: FILE OF REAL;
    MISC: FILE OF INTEGER;

PROCEDURE PREFINISH;
BEGIN
  OKSET:=['Y','y','N','n'];
  GOTOXY(0,0);CRT(ERASEOS);
  WRITELN;
  WRITELN('  MEMORY AVAILABLE IN WORDS: ',MEMAVAIL);
  WRITELN;
  WRITELN('  MOTION ANALYSIS PROGRAM OPTIONS : ');
  WRITELN;
  WRITELN('  DO YOU WISH TO CONTINUE THIS GAME LATER (Y/N)? ');
  WRITELN;WRITELN;
  WRITE('  ');    GFLAG1:=GETCHAR(OKSET+[CHR(13)]);
  CRT(ERASEOS); CRT(ERASEOS);
  IF GFLAG1 IN ['Y','y'] THEN SAVEFLAG:=TRUE ELSE SAVEFLAG:=FALSE;
END; ( PREFINISH )

PROCEDURE START;
BEGIN
  GOTOXY(0,0);CRT(ERASEOS);
  WRITELN;
  WRITELN('  MEMORY AVAILABLE IN WORDS: ',MEMAVAIL);
  WRITELN;
  WRITELN('  MOTION ANALYSIS PROGRAM OPTIONS : ');
  WRITELN(' ');
  WRITELN('  0  Motion Analysis Program');
  WRITELN('  1  Rerun Motion Analysis Program');
  WRITELN('  2  Build or Modify Data Base');
  WRITELN('  3  Terminate Program');
  WRITELN;
  WRITE('  ');    GFLAG:=GETCHAR(LEVEL0);
  CRT(ERASEOS); CRT(ERASEOS)
END; ( START )


PROCEDURE SAVIT;
VAR I,J:INTEGER;
BEGIN
  REWRITE(BNOW, '#5:OLD1.DATA');
  FOR J:=1 TO MAXBLUE DO BEGIN
    BNOW^:=BXY[1,J]; PUT(BNOW);
    BNOW^:=BXY[2,J]; PUT(BNOW);
    BNOW^:=GCUS[J]; PUT(BNOW);
    BNOW^:=GSPD[J]; PUT(BNOW);
  END;
  CLOSE(BNOW,LOCK);
  REWRITE(BOLD, '#5:OLD2.DATA');
  FOR J:=1 TO MAXBLUE DO BEGIN
```

```
    BOLD^:=OLDBXY[1,J]; PUT(BOLD);
    BOLD^:=OLDBXY[2,J]; PUT(BOLD);
  END;
  CLOSE(BOLD,LOCK);
  REWRITE(ONOW, '#5:OLD3.DATA');
  FOR J:=1 TO MAXORANGE DO BEGIN
    ONOW^:=OXY[1,J]; PUT(ONOW);
    ONOW^:=OXY[2,J]; PUT(ONOW);
  END;
  CLOSE(ONOW,LOCK);
  REWRITE(OOLD, '#5:OLD4.DATA');
  FOR J:=1 TO MAXORANGE DO BEGIN
    OOLD^:=OLDOXY[1,J]; PUT(OOLD);
    OOLD^:=OLDOXY[2,J]; PUT(OOLD);
  END;
  CLOSE(OOLD,LOCK);
  REWRITE(MISC, '#5:OLD5.DATA');
  MISC^:=TIME; PUT(MISC);
  MISC^:=BLUEMOVE; PUT(MISC);
  MISC^:=ORANGEMOVE; PUT(MISC);
  CLOSE(MISC,LOCK);
END;

BEGIN  ( TMAFINIT )
  CLOSE(BLUHIST,LOCK);
  CLOSE(BLUGAME,LOCK);
  CLOSE(ORNHIST,LOCK);
  CLOSE(ORNGAME,LOCK);
  CLOSE(ATIME,LOCK);
  LEVEL0:=['0','1','2','3'];
  QUITFLAG:=FALSE;
  RRFLAG:=FALSE;
  START;
  CASE GFLAG OF
    '0' : CASE0:=TRUE;
    '1' : CASE1:=TRUE;
    '2' : CASE2:=TRUE;
  END;  ( CASE )
  IF GFLAG = '3' THEN BEGIN
    PREFINISH;
    IF SAVEFLAG THEN SAVIT;
    STOPFLAG:=TRUE;
  END;
END; ( TMAFINIT )

(*$S+*)
UNIT GETCRT; INTRINSIC CODE 23 DATA 24;

INTERFACE
  TYPE CRTCOMMAND=(ERASEOS,ERASEOL,UP,DOWN,RIGHT,LEFT,LEADIN);
```

```
        SETOFCHAR=SET OF CHAR;

    VAR  CRTINFO:   PACKED ARRAY[CRTCOMMAND] OF CHAR;
         PREFIXED:  ARRAY[CRTCOMMAND] OF BOOLEAN;
         F:FILE;
         PROCEDURE GETCRTINFO;
         PROCEDURE CRT(C : CRTCOMMAND);
         PROCEDURE PROMPTAT(Y: INTEGER; S: STRING);
         FUNCTION GETCHAR(OKSET: SETOFCHAR): CHAR;
         PROCEDURE GETSTRING(VAR S: STRING; MAXLEN: INTEGER);
         PROCEDURE GETINTEGER (VAR NUMBER,LEN : INTEGER);

IMPLEMENTATION
    CONST
         BS = 8;
         SPACE = 32;
         CR = 13;
    VAR  BUFFER: PACKED ARRAY[0..511] OF CHAR;
         I,BYTE: INTEGER;
         CH: CHAR;
         GOOD: BOOLEAN;
         S1:STRING[1];
         STEMP: STRING[80];
         POSITION: INTEGER;
         CHARRAY: ARRAY[1..10] OF CHAR;
         READINTEGER: INTEGER;
         DIGITS,OKSET: SET OF CHAR;
         OLDNBR: INTEGER;
         OLD: BOOLEAN;


PROCEDURE GETCRTINFO;
{                                                                    }
{     READ SYSTEM.MISCINFO AND GET CRT CONTROL CHARACTER INFO        }
{                                                                    }
BEGIN
  RESET(F,'*SYSTEM.MISCINFO');
  I:=BLOCKREAD(F,BUFFER,1);
  CLOSE(F);
  BYTE:=ORD(BUFFER[72]); {  PREFIX INFORMATION BYTE  }
  CRTINFO[LEADIN]:=BUFFER[62];   PREFIXED[LEADIN]:=FALSE;
  CRTINFO[ERASEOS]:=BUFFER[64];  PREFIXED[ERASEOS]:=ODD(BYTE DIV 8);
  CRTINFO[ERASEOL]:=BUFFER[65];  PREFIXED[ERASEOL]:=ODD(BYTE DIV 4);
  CRTINFO[RIGHT]:=BUFFER[66];    PREFIXED[RIGHT]:=ODD(BYTE DIV 2);
  CRTINFO[UP]:=BUFFER[67];       PREFIXED[UP]:=ODD(BYTE);
  CRTINFO[LEFT]:=BUFFER[68];     PREFIXED[LEFT]:=ODD(BYTE DIV 32);
  CRTINFO[DOWN]:=CHR(10);        PREFIXED[DOWN]:=FALSE;
END;

PROCEDURE CRT;
```

```
{                                                                    }
{     CRT COMMANDS ARE : ERASEOS,ERASEOL,UP,DOWN,RIGHT.LEFT.         }
{                                                                    }
BEGIN
  IF PREFIXED[C] THEN UNITWRITE(1,CRTINFO[LEADIN],1,0,12);
  UNITWRITE(1,CRTINFO[C],1,0,12);
END;

PROCEDURE PROMPTAT;
BEGIN
  GOTOXY(0,Y);WRITE(S);CRT(ERASEOL);
END;


FUNCTION GETCHAR;
{                                                                }
{   GET A CHARACTER, BEEP IF NOT IN OKSET, ECHO ONLY IF PRINTING  }
{                                                                }
BEGIN
  REPEAT
    READ(KEYBOARD,CH);
    IF EOLN(KEYBOARD) THEN CH:=CHR(13);
    GOOD:= CH IN OKSET;
    IF NOT GOOD THEN WRITE(CHR(7))
      ELSE IF CH IN [' '..'~'] THEN WRITE(CH);
  UNTIL GOOD;
  GETCHAR:=CH;
END;


PROCEDURE GETSTRING;
{                                                              }
{   GET AND ECHO A STRING UP TO MAXLEN CHARS LONG.             }
{   IF NULL STRING ENTERED, DEFAULT AND PRINT PREVIOUS VALUE.  }
{                                                              }
BEGIN
  OKSET:=[' '..'~'];
  S1:=' ';
  STEMP:='';
  FOR POSITION := 1 TO MAXLEN DO
     WRITE('_');
  POSITION := 1;
  FOR POSITION := 1 TO MAXLEN DO
     WRITE(CHR(BS));
  POSITION :=1;
  REPEAT
    IF LENGTH(STEMP) = 0 THEN S1[1]:=GETCHAR(OKSET + [CHR(13)])
      ELSE IF LENGTH(STEMP)=MAXLEN THEN S1[1]:=GETCHAR([CHR(13),CHR(8)])
            ELSE S1[1]:=GETCHAR(OKSET + [CHR(13),CHR(8)]);
    IF S1[1] IN OKSET THEN STEMP:=CONCAT(STEMP,S1)
```

```
      ELSE IF S1[1]=CHR(8) THEN
        BEGIN
          CRT(LEFT); WRITE('_'); CRT(LEFT);
          DELETE(STEMP,LENGTH(STEMP),1);
        END;
  UNTIL S1[1] = CHR(13);
  IF LENGTH(STEMP) <> 0 THEN S:=STEMP
  ELSE WRITE(S);
END;

PROCEDURE GETINTEGER;
  BEGIN
    OLD := FALSE;
    OLDNBR:=NUMBR;
    DIGITS:=['0'..'9'];
    FOR POSITION := 1 TO LEN DO
      WRITE('_');
    FOR POSITION := 1 TO LEN DO
      WRITE(CHR(BS));
    POSITION := 1;
    WHILE POSITION = 1 DO
      BEGIN
        READ(KEYBOARD,CHARRAY[POSITION]);
        IF(CHARRAY[1] IN [CHR(SPACE),CHR(CR)]) THEN
           BEGIN
             NUMBR := OLDNBR;
             WRITE(NUMBR);
             OLD := TRUE;
             POSITION := POSITION+1;
           END
        ELSE
           IF(CHARRAY[1] IN DIGITS) THEN
           BEGIN
             WRITE(KEYBOARD,CHARRAY[POSITION]);
             POSITION:=POSITION + 1;
           END
           ELSE
             WRITE(CHR(7));
        END;
      IF NOT OLD THEN
      BEGIN
      WHILE POSITION<=LEN DO
        BEGIN
          READ(KEYBOARD,CHARRAY[POSITION]);
          IF (CHARRAY[POSITION] IN DIGITS) THEN
            BEGIN
              WRITE(CHARRAY[POSITION]);
              POSITION := POSITION + 1;
            END
          ELSE IF CHARRAY[POSITION] = CHR(BS) THEN
```

```
          BEGIN
            IF POSITION > 1 THEN
              BEGIN
                WRITE(CHR(BS));
                WRITE('_');
                WRITE(CHR(BS));
                POSITION := POSITION - 1
              END;
            IF POSITION <= 1 THEN POSITION := 1;
          END
        ELSE
          BEGIN
            IF (CHARRAY[POSITION] IN [CHR(SPACE),CHR(CR)]) THEN
              POSITION:=LEN + 1
            ELSE
              WRITE(CHR(7))
          END; ( begin )
        END  ( if-then-else )
      END;
      READINTEGER:=0;
      FOR POSITION:=1 TO LEN DO
        BEGIN
          IF (CHARRAY[POSITION] IN DIGITS) THEN
            READINTEGER:=10*READINTEGER+ORD(CHARRAY[POSITION])-ORD('0');
        END;
      IF OLD THEN NUMBR:=OLDNBR ELSE
      NUMBR:=READINTEGER;
    END;
  BEGIN
  END.

(*$S+*)
UNIT THESIS0; INTRINSIC CODE 25 DATA 26;

INTERFACE
  USES TURTLEGRAPHICS;

  VAR
      SHIPFIG: PACKED ARRAY[1..8,1..7] OF BOOLEAN;
      BXY,OXY: PACKED ARRAY[1..2,1..10] OF REAL;
      SCALE: REAL;
      XCTR,YCTR,MAXBLUE,MAXORANGE: INTEGER;

      PROCEDURE NUMBER(X,Y,J: INTEGER);
      PROCEDURE LETTER(X,Y,J: INTEGER);
      PROCEDURE BOXSTRING(X,Y: INTEGER; S: STRING);
      PROCEDURE CLEARBOX;
      PROCEDURE DOWNSCALE;
      PROCEDURE UPSCALE;
```

```
IMPLEMENTATION

  CONST
       XC = 140;
       YC = 96;

   VAR
       VALUE: INTEGER;
       NUM,XT,YT: INTEGER;
       VAL:  CHAR;
       BX,BY,CX,CY,J: INTEGER;
       L,R,B,T: INTEGER;

PROCEDURE NUMBER;
BEGIN
  NUM:=J-1;
  XT:=X-3;
  YT:=Y-9;
  MOVETO(XT,YT);
  VAL:=CHR(NUM+ORD('0'));
  WCHAR(VAL);
END;

PROCEDURE LETTER;
BEGIN
  NUM:=J-1;
  XT:=X-3;
  YT:=Y-9;
  MOVETO(XT,YT);
  VAL:=CHR(NUM+ORD('A'));
  WCHAR(VAL);
END;

PROCEDURE BOXSTRING;
BEGIN
  PENCOLOR(NONE);MOVETO(X,Y);
  L:=X-3; R:=X+2+7*LENGTH(S);
  B:=Y-2; T:=Y+10;
  VIEWPORT(L,R,B,T);
  FILLSCREEN(BLACK);
  VIEWPORT(0.279,0.191);
  PENCOLOR(NONE); MOVETO(L,B);
  PENCOLOR(WHITE); MOVETO(L,T);
  MOVETO(R,T);MOVETO(R,B);MOVETO(L,B);
  PENCOLOR(NONE); MOVETO(L+3,B+2);
  WSTRING(S);
END;

PROCEDURE CLEARBOX;
BEGIN
```

```
  PENCOLOR(NONE);
  L:=0; B:=0; R:=5 + 7 * 40; T:=12;
  VIEWPORT(L,R,B,T);
  FILLSCREEN(BLACK);
  VIEWPORT(0,279,0,191);
END;

PROCEDURE DOWNSCALE;
BEGIN
  SCALE:=SCALE / 2;
  INITTURTLE;
  PENCOLOR(NONE);
  IF MAXBLUE > 0 THEN BEGIN
    FOR J:= 1 TO MAXBLUE DO BEGIN
      BX:=ROUND((BXY[1,J]-XCTR)/SCALE)+XC;
      BY:=ROUND((BXY[2,J]-YCTR)/SCALE)+YC;
      DRAWBLOCK(SHIPFIG,2,0,0,7,3,BX,BY,10);
      NUMBER(BX,BY,J);
    END;
  END;
  IF MAXORANGE > 0 THEN BEGIN
    FOR J:= 1 TO MAXORANGE DO BEGIN
      OX:=ROUND((OXY[1,J]-XCTR)/SCALE)+XC;
      OY:=ROUND((OXY[2,J]-YCTR)/SCALE)+YC;
      DRAWBLOCK(SHIPFIG,2,0,0,7,3,OX,OY,5);
      LETTER(OX,OY,J);
    END;
  END;
END;

PROCEDURE UPSCALE;
BEGIN
  SCALE:=SCALE * 2;
  INITTURTLE;
  PENCOLOR(NONE);
  IF MAXBLUE > 0 THEN BEGIN
    FOR J:= 1 TO MAXBLUE DO BEGIN
      BX:=ROUND((BXY[1,J]-XCTR)/SCALE)+XC;
      BY:=ROUND((BXY[2,J]-YCTR)/SCALE)+YC;
      DRAWBLOCK(SHIPFIG,2,0,0,7,3,BX,BY,10);
      NUMBER(BX,BY,J);
    END;
  END;
  IF MAXORANGE > 0 THEN BEGIN
    FOR J:= 1 TO MAXORANGE DO BEGIN
      OX:=ROUND((OXY[1,J]-XCTR)/SCALE)+XC;
      OY:=ROUND((OXY[2,J]-YCTR)/SCALE)+YC;
      DRAWBLOCK(SHIPFIG,2,0,0,7,3,OX,OY,5);
      LETTER(OX,OY,J);
    END;
```

```
  END;
END;

BEGIN
END.
```

LIST OF REFERENCES

1. Simpson, H., "A Human-Factors Style Guide for Program Design", Byte, v. 7, p. 108-132, April 1982.

2. Rocke, J. W., "Format Your Inputs, the Natural Way", Personal Computing, v. 5, p. 67-70, November 1981.

BIBLIOGRAPHY

Andrus, A., Prototype Programs for an Interactive Gaming Graphics Plotting Capability, Naval Postgraduate School Report NPS55-81-009, February 1981.

Apple Computer Inc., Apple Pascal: Language Reference Manual, 1980.

Apple Computer Inc., Apple Pascal: Operating System Reference Manual, 1980.

Grogono, P., Programming in Pascal: Revised Edition, Addison-Wesley, 1980.

Jensen, K. and Wirth, N., Pascal: User Manual and Report, 2nd ed., Springer-Verlag, 1974.

Leuhrmann, A. and Peckham, H., Apple Pascal: A Hands-on Approach, McGraw-Hill, 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Office of Naval Research, Code 230 Attn: Mr. R. Nagelhout Arlington, Virginia 22217	1
4.	Prof. Alvin F. Andrus, Code 55As Naval Postgraduate School Monterey, California 93940	2
5.	Prof. James D. Esary, Code 55Ey Naval Postgraduate School Monterey, California 93940	2
6.	Lt. Jack W. McCorkle, USN 466 Longtowne Court Glen Burnie, Maryland 21061	2

Th
MI
c.
Thesis 199117
M18193 McCorkle
c.1 Tactical motion
analyzer (TMA).
14OCT83 27914

Thesis 199117
M18193 McCorkle
c.1 Tactical motion
analyzer (TMA).

thesM18193
Tactical motion analyzer (TMA).
3 2768 000 98295 3
DUDLEY KNOX LIBRARY